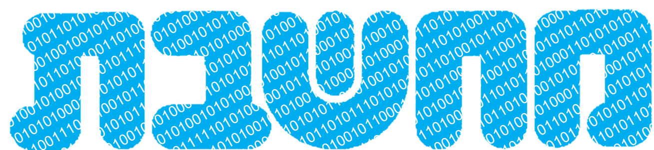


צעדים ראשונים בתכנות באסמבלי

ATARI 8

600XL 800XL ATARI 400 and 800

ATARI
600XL, 800XL, 400, 800

מחשבת

צעדים ראשונים בתכנות באסמבלי

יחידה 8

אסמבלי ושפת מכונה

ASSEMBLY & MACHINE)
(LANGUAGE

תוכן העניינים

הקדמת המחבר

לאחר הצלחתה המסחררת של "מחשבת 7", הסתערתי במלוא המרץ על מלאכת כתיבת יחידת הלימוד הבאה שעניינה: DISPLAY LIST או בתרגום לעברית: "מתווה התצוגה". DISPLAY LIST הוא אחד הנושאים המרתקים ביותר במסע אל זכרונו של המחשב, והוא היסוד לבנייתם של משחקים ושל תוכנות גרפיות למיניהם במחשב האטארי.

דא עקה, כבר בשלבי הכתיבה הראשונים נתקלתי במחסום בלתי עביר – הסתבר לי כי הכלים הזמינים בבייסיק פשוט אינם מאפשרים להתקדם הלאה. בלית ברירה, יצאתי לחפש כלים נוספים חדשים, ועד מהירה מצאתי את עצמי ניצב מול המילים המפחידות: "אסמבלר" ו"שפת מכונה".

האם על מנת "לדבר" אל אותה מכונה, הרהרתי בחשש, תידרש ממני איזו גאונות תכנותית יוצאת דופן? או שמא מומחיות של טכנאי מחשבים? ומאחר שאין בי לא מזה ולא מזה, ניגשתי למשימה עם לא מעט היסוס. תחילה קראתי כמה ספרים בנושא וגם עברתי על כמה וכמה שורות קוד באסמבלי, ואט אט התחלתי לארגן את הדברים ולרשום את הידע אשר התחלתי לצבור.

במהלך כתיבת חוברת זו, הצבתי לעצמי שלושה עקרונות והתחייבתי לפעול על פיהם: ראשית, **פשטות** – כלומר להקצות לכל פקודה שאדון בה, משימה בודדת אחת, קלה ככל האפשר, ואת המספר מינימלי הנדרש של שורות קוד. שנית, **לינאריות** – כלומר, להתקדם בשלבים, פקודה אחר פקודה, ולא להתפתות לדלג בין עניינים שונים או להכניס לדיון נושאים שאינם קשורים במישרין לתחום עיסוקה של חוברת זו. ולבסוף, **הנגשה** – כלומר להציג את עקרונות השפה ואת פקודותיה בדרך המובנת ביותר לא רק למתכנתים, ולשם כך, להרבות בדוגמאות מתחומים אחרים שמחוץ לעולם המחשב.

עם סיום כתיבתה של חוברת זו, אני חש סיפוק רב. שפת האסמבלי ושפת המכונה אשר נראו לי תמיד תחום עיסוקם של יודעי ח"ן – הפכו נגישים וברי השגה. אמנם, יחידת לימוד זו לא הפכה אותי – וכנראה גם לא את קוראיה – למפתחי משחקים בשפת מכונה, אך אני סבור שהיא יכולה לשמש בסיס טוב להמשך לימוד והעמקה בתחום.

ולבסוף, לאחר שהעשרתי את סט הכלים הזמינים לעבודה, אוכל לחזור עתה אל הנושא שבו פתחתי – מתווה התצוגה (DISPLAY LIST) ולהשלים בקרוב, גם את כתיבת "מחשבת 9".

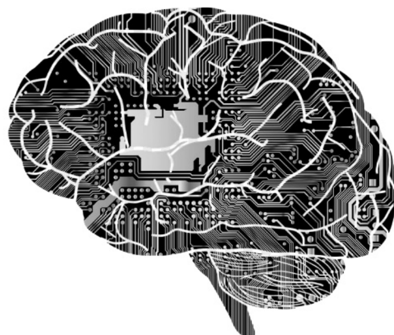
נדב שטכמן פולישוק

אפריל 2020

מבוא

קום לרגע מהכיסא וחלץ את איבריך לאחר הישיבה ממושכת מול מסך הטלוויזיה...

עתה חזור והתיישב על הכיסא – אך בטרם תעשה כן – חשוב לרגע אילו הוראות נדרש המוח למסור לאיברי גופך כדי לגרום לאותה "מכונה" לעבור ממצב עמידה למצב ישיבה. המוח שלך, בדומה למעבד המחשב, מבין ומבצע הוראות, וכאשר מוסרים לו את ההוראה "שב", הוא יודע לתרגם אותה לסדרה של הוראות פשוטות יותר וישירות לביצוע – הוראות מסוג: כווץ שריר מסוים, הרפה שריר אחר, וכולי – כל זאת על מנת להביא את הגוף שלך למצב ישיבה.



את ההוראה "שב" אפשר לדמות לפקודה בשפת הבייסיק. בדיוק כשם שהמוח שלך יודע לתרגם את הפקודה "שב" לסדרה של הוראות פשוטות לביצוע, כך יודע המחשב לתרגם את פקודות הבייסיק השונות (כגון: **PRINT**, **GOTO**) לסדרה של הוראות פשוטות לביצוע על ידי המעבד.

נתבונן לרגע על פקודת הבייסיק "**PRINT**". אם תפקוד על המחשב: **PRINT 5**, הוא יכתוב כמובן על המסך את הספרה 5. גם אתה, אם תקבל את ההוראה: "כתוב את הספרה 5", תגיב בנטילת העט וכתיבת הספרה 5. (בהנחה, כמובן, שאתה מציית להוראות). ניתן לסכם ולומר כי יש דמיון בין שפת התכנות בייסיק לבין השפה הטבעית (אנגלית, עברית וכולי).

שפת תכנות המשתמשת באלמנטים תחביריים המזכירים שפה טבעית נקראות "שפת תכנות עילית" (High-level programming language). הבייסיק היא שפת תכנות עילית, וכמוה גם שפות תכנות אחרות כמו: פסקל ו-C.

עכשיו נשאלת השאלה כיצד מתבצע התרגום מפקודה בשפה עילית לרשימת הוראות לביצוע. בכל הנוגע לפעילות המוח, אין בידיי תשובה לשאלה כיצד יודע המוח לתרגם את הפקודה "שב" לסדרה של הוראות לשרירי גופך. זו שאלה שיש להפנות לנוירולוגים (וספק אם יש בידיהם תשובה ממצה)...

בתחום המחשב, בכל אופן, התשובה פשוטה הרבה יותר. המחשב יודע לתרגם את הפקודות "העיליות" הנמסרות לו לסדרת הוראות לביצוע והוא עושה זאת באמצעות תוכנת מחשב המכונה "מהדר" (COMPILER).

סדרה זו של "הוראות לביצוע" – אשר נמסרת למעבד המחשב – כתובה ב"שפת מכונה". שפת המכונה היא השפה **היחידה** שמעבד המחשב יודע לקרוא והוא מבצע את הוראותיה באופן ישיר ובמהירות הבזק.

כיצד נראית "שפת מכונה"? ובכן, כל אחת מן הפקודות של שפת המכונה בנויה מרצפים של סיביות (Bits) – כלומר מסדרה בינארית של הספרות 0 ו-1.

ניקח לדוגמה את רצף הסיביות הבא:

1010100100100010100011011100100000000010

הרצף המדויק הזה – ככל שיימסר למעבד המחשב שלך – יגרום לו לצבוע את רקע מסך הטלוויזיה שלך באדום!

? האם היית מסוגל לכתוב תכניות שלמות בשפת מכונה (כלומר לזכור ולהקליד רצפים ארוכים של יחידות 0 ו-1)? סביר להניח שלא. בדיוק לשם

כך באה לעולם שפת אסמבלי – על מנת להפוך את התכנות בשפת המכונה למעט יותר "אנושי".

שפת האסמבלי מוגדרת כ"שפת סף" – היא אמנם אינה "שפה עילית", אך היא גם אינה שפה של אפסים ואחדים. השימוש בשפת האסמבלי חוסך ממך את השימוש המייגע ברצפים הבינאריים האינסופיים. במקום כל רצף כזה של אפסים ואחדים, מאפשרת לך שפת האסמבלי להקליד רצף קצר של אותיות, שקל לזוכרו.

אם נחזור שוב לרצף שהצגנו למעלה. בשפת אסמבלי ייכתב הרצף הזה בשפה מעט יותר מובנת:

```
LDA   #$22
STA   $2C8
```

אל דאגה, כעת אינך נדרש להבין את משמעות האותיות והסימנים הללו. עוד נעמוד על כל אחד מהם בהמשך. הבאתי את הדוגמה הזו רק כדי להמחיש לך את ההבדל בין "שפת מכונה" ו"שפת סף".

? מהם יתרונותיה של שפת האסמבלי (או שפת המכונה) על פני שפות עיליות?

ובכן, בראש ובראשונה זוהי המהירות. כאשר אתה כותב שורות קוד בשפת הבייסיק ומורה למחשב לבצע אותן, המעבד משתמש בהרבה משאבים – הוא נדרש לתרגם כל פקודה לרצף (לפעמים ארוך מאוד) של הוראות בשפת מכונה ורק אז ניגש לבצע אותם. ככל שהתוכנות שתכתוב בבייסיק (או בשפות תכנות עיליות אחרות) ארוכות יותר ומורכבות יותר, כך זמן הביצוע שלהן יארך.

זאת ועוד, בעוד ששפת הבייסיק (כמו גם שפות עיליות רבות אחרות) משתמשת בשיטת הספירה העשרונית (אשר כמובן מובנת יותר לבני אדם), שפת האסמבלי משתמשת בשיטת ספירה אחרת – בשיטה ההקסה-דצימלית (בסיס 16). שיטת הספירה ההקסה-דצימלית – שעוד ארחיב עליה את הדיבור בהמשך – "טבעית" יותר למעבד של המחשב, והחישובים השונים שהוא מבצע בסביבת עבודה זו מהירים הרבה יותר מחישובים בשיטה העשרונית.

ולבסוף, ישנן הוראות שניתן לבצען אך ורק באסמבלי, ובמובן זה שפת הבייסיק אינה יכולה להתחרות עמה. למעשה, השימוש שעשית עד כה (במחשבת 6, 7) בפניות ישירות לזיכרון (באמצעות פקודת הבייסיק: **POKE**) גם הוא נחשב כהוראות ישירות לביצוע – ממש כמו הוראות של שפת מכונה. ודאי הבנת כבר כי הרבה דברים אינך יכול לבצע ללא פניות ישירות אלו אל הזיכרון.

בהמשך החוברת – אחרי שתרכוש את המיומנות בשפת האסמבלי – תגלה דברים חדשים ומפתיעים שתוכל לבצע במחשב שלך – דברים שלא תוכל לבצע בבייסיק (אפילו לא בשימוש בפקודה: **POKE**).

? ומהם החסרונות של השימוש בשפת האסמבלי?

ובכן, ראשית כל זהו אורכן של התוכנות. בעוד שהתכנות (האלגנטי) בשפה עילית מבוסס על מספר מצומצם של שורות קוד – הרי ששפת האסמבלי מבוססת על הוראות קצרות ופשוטות ביותר ועל כן נדרשים לפעמים עשרות הוראות שכאלה על מנת לבצע משימה "פשוטה". היזכר שוב בדוגמה שבה פתחנו את המבוא. ברור שההוראה: "שב" פשוטה וקצרה הרבה יותר מרצף של הוראות לכל אחד משרירי הגוף הרלוונטיים להתכווץ או להתרפות.

ושנית, ישנה מידה לא מעטה של סרבול שתידרש לו במלאכת התכנות על מנת לבצע פעולות שהשפה העילית מבצעת בקלות רבה מאוד. דע לך כי התכנות בשפה עילית כמו בייסיק חוסך ממך לטפל בעניינים רבים שנראים

לך טריוויאליים וברורים מאליהם – כמו למשל ניהול הזיכרון. באופן פשטני ניתן לסכם ולומר כי התכנות בבייסיק מספק לך את כל הגלגלים ואילו התכנות בשפת אסמבלי מחייב אותך להמציא כל פעם את הגלגל מחדש.

אך על דאגה, לפני שנצא לדרך, אדגיש כי פנינו אינן להיפרד לשלום מסביבת הבייסיק. ככלל, נמשיך לכתוב את התוכניות שלנו בבייסיק וניעזר בשפת האסמבלי כדי להעשיר אותן ולהוסיף להן פונקציות חדשות.

בפרק הבא נלמד להשתמש בכתבן הייחודי של שפת האסמבלי – הנקרא: "אסמבלר". האסמבלר יאפשר לנו לכתוב את שורות הקוד שלנו באסמבלי, ולהמיר אותן אחר כך לתוכנות בשפת מכונה.

אך לפני כן – הערה גרפית. במהלך הספר הקפדתי על הבחנה גרפית ברורה בין פקודות של שפת האסמבלי – הצבועות בירוק – לבין פקודות שאינן חלק מן השפה.

פרק א
הכרת ממשק העבודה של האסמבלר

כתבן האסמבלר אשר נשתמש בו לאורך יחידת לימוד זו הוא:
Atari 400/800 Assembler Cartridge CXL4003.

אם יש ברשותך את הקרטריג' עצמו, הכנס אותו למקום המיועד לכך במחשב האטארי שלך. לכונן הדיסקטים, הכנס דיסקט עם מערכת ההפעלה DOS והפעל את המחשב במצב: BASIC

אם אתה עובד על אמולטור (תוכנת מחשב המדמה את פעולתו של מחשב האטארי), היכנס לתפריט File, בחר באפשרות: Attach Cartridge וטען את הקובץ: Assembler Editor.rom (המדמה את הקרטריג'): (השאר את ברירת המחדל כך: Standard 8kb Cartridge). ודא כי האמולטור פועל במצב BASIC וטען קובץ ATR (המדמה דיסקט) אשר כולל את מערכת ההפעלה DOS.

לאחר שהמחשב יאתחל את עצמו תופיע המילה EDIT ותוכל להתחיל לעבוד.

בכל שלב תוכל להפסיק את עבודתך באסמבלר, אם תשלוף החוצה את הקרטריג', או אם תיכנס באמולטור לתפריט File ותבחר באפשרות: Detach Cartridge

אל תשכח לשמור את התוכניות שכתבת לפני הסרת הקרטריג', וזאת משום שהסרת הקרטריג' תאתחל באופן מידי את המחשב.

פקודות הכתבן

אפתח בהבהרה: בפרק זה נעסוק אך ורק בפקודות של **כתבן** האסמבלר. פקודות אלו ישמשו אותך כדי לתפעל את הכתבן אך זכור כי הן **אינן** חלק משפת התכנות "אסמבלי", ובוודאי שאינן מתורגמות לשפת מכונה.

אני ממליץ לך להתנסות בכל אחת מפקודות הכתבן שאציג לפניך. לשם כך, תוכל להקליד שורות טקסט סתמיות ללא חשש. כל עוד לא תפקוד על המחשב לתרגם אותן לשפת מכונה, לא יתרחש דבר.

הפקודה **NUM**

התכנות באסמבלי – בדומה לתכנות בבייסיק – מחייב כי כל שורת קוד תפתח במספר שורה.

תוכל להקליד בעצמך את מספרי השורות ותוכל גם להפעיל את המספור האוטומטי. לשם כך, הקלד את הפקודה **NUM** והקש ENTER.

ברירת המחדל היא מספור אוטומטי בסדר עולה: 10, 20, 30...

תוכל גם לשנות את המרווח בין מספרי השורות. כך למשל, אם תקליד את הפקודה: **NUM20**, מספרי השורות יתקדמו בסדר הבא: 20, 40, 60...

ולבסוף, אם תקליד את הפקודה **NUM20,5**, יתחיל מספור השורות בשורה 20, אך מרווח ההתקדמות יהיה 5, כלומר מספרי השורות יהיו: 20, 25, 30...

לחיצה על מקש ENTER – מבלי להקליד אף תו בשורה – תפסיק את המספור האוטומטי ותאפשר לך להמשיך ולהקליד בעצמך את מספרי השורות הבאות.

הפקודה **REN**

כתבן האסמבלר מתייחס, כאמור, רק לשורות קוד הפותחות במספר, והוא עוקב אחר שורות הקוד לפי סדר מספורם.

במהלך כתיבת התוכנית, תוכל כמובן להוסיף שורת קוד נוספת בין שתי שורות קיימות, וזאת אם תמספר אותה בהתאם. כך לדוגמה תוכל להוסיף שורה שמספרה 23 בין שורות 20 ו-30.

הפקודה **REN** מאפשרת לך לשמור על סדר השורות הקיים בתוכנה שכתבת, אך למספר מחדש את כולן. כל שעליך לעשות הוא להקליד את הפקודה: **REN** ולהקיש: ENTER.

בדומה לפקודה **NUM** אם תקליד את הפקודה: **REN30**, מספרי השורות יתקדמו בסדר הבא: 30, 60, 90...

ואם תקליד את הפקודה **REN30,20**, יתחיל מספור השורות בשורה 30, ומרווח ההתקדמות יהיה 20, כלומר מספרי השורות יהיו: 30, 50, 70...

הפקודה **FIND**

פקודה זו יכולה לסייע לך לאתר ביטוי שמופיע איפשהו בתוך שורות הקוד.

מבנה הפקודה הוא זה:

FIND/ביטוי/

כך למשל, על מנת לאתר את המופע **הראשון** של הביטוי **JMP** פקוד:

FIND/JMP/

אם תרצה שהכתבן יציג את **כל** שורות הקוד בהן מופיע הביטוי שאתה מחפש, שנה את הפקודה כך:

```
FIND/JMP/,A
```

הפקודה LIST

הפקודה **LIST** דומה מאוד לפקודה המקבילה המוכרת לך בבייסיק. היא מציגה את כל שורות הקוד לפי סדרן המספרי.

אם תרצה לראות רק את שורה מספר 50, לדוגמה, פקוד:

```
LIST 50
```

ואם תרצה לראות את טווח השורות שבין שורה 50 לשורה 100, הקלד:

```
LIST 50,100
```

הפקודה **LIST** משמשת גם לשמירת שורות הקוד על גבי דיסקט.

על מנת לשמור את שורות הקוד, הקלד:

```
LIST #D:FileName.ASM
```

שים לב לצורת הפנייה לאמצעי הקלט/פלט, אשר שונה מהצורה שהתרגלת אליה בסביבת העבודה של בייסיק!

בשלב זה, ראוי להזכיר גם הוראות שגיאה שאתה עלול להיתקל בהן אגב עבודתך עם אמצעי הקלט/פלט. כך למשל, אם אמצעי הקלט/פלט שתנסה להשתמש בו לא קיים (או אינו מחובר כהלכה), תתקבל הודעת שגיאה: ERROR 130

אם שם הקובץ שתקליד לא תקין (למשל מכיל סימנים גרפיים), תתקבל הודעת שגיאה: ERROR 165

הפקודה ENTER

הפקודה **ENTER** משמשת על מנת לקרוא את שורות הקוד אשר שמרת על גבי הדיסקט. צורת הפקודה היא:

```
ENETR #D:FileName.ASM
```

גם כאן אתה עלול להיתקל בהודעות שגיאה אגב שימוש לא נכון באמצעי הקלט/פלט. כך למשל, אם תנסה לקרוא קובץ שאינו קיים, תתקבל הודעת שגיאה: ERROR 170

הפקודה DEL

על מנת למחוק שורה שהקלדת תוכל – ממש כפי שהתרגלת בבייסיק – להקליד את מספרה ולהקיש ENTER.

ואולם כתבן האסמבלר מאפשר לך למחוק שורות בצורה נוספת. על מנת למחוק את מספר 50, לדוגמה, תוכל לפקוד:

```
DEL 50
```

אם ברצונך למחוק את כל השורות בטווח שבין 50 ל-100, תוכל להקליד:

```
DEL 50,100
```

שים לב, אם תבקש למחוק באמצעות הפקודה **DEL** שורה שלא קיימת, תתקבל הודעת השגיאה: ERROR 2

הפקודה NEW

הפקודה **NEW** – בדומה למקבילתה בבייסיק – תמחק את כל שורות הקוד.

הפקודה ASM

הפקודה **ASM** היא לב לבו של האסמבלר. פקודה זו מתרגמת את שורות הקוד באסמבלי לפקודות המקבילות להן בשפת מכונה – אשר אותה ורק אותה, כאמור, יוכל המעבד לקרוא.

כאשר תפקוד על המחשב את הפקודה **ASM**, יקרו למעשה שני דברים:

1. ראשית, כל פקודה באסמבלי תתורגם למקבילה הבינארית שלה בשפת המכונה. בנוסף, יופיעו על המסך, בזה אחר זה, הפקודות המספריות אשר תורגמו לשפת מכונה, כמובן לפי סדר הופעתן בשורות הקוד. שים לב: לצורך הנוחות, המסך לא יתמלא ברצפים ארוכים של אפס ואחד, אלא לתרגומם המספרי לפי השיטה ההקסה-דצימלית.

   אם נחזור שוב לדוגמה שהצגנו בפרק המבוא, הרי שלאחר הקלדת הפקודה **ASM** נצפה למצוא, במקום הרצף הבינארי: 1010100100100010100011011100100000000010, את הרצף המספרי:
   A9 22 8D C8 02

2. אם שורות הקוד באסמבלי נכתבו כהלכה, התוכנית המקבילה (המתורגמת לשפת מכונה) תישמר בתוך מקטע הזיכרון שהוקצה עבורה מבעוד מועד.

אם שורת הקוד באסמבלי לא נכתבה כהלכה, תתקבל הודעת השגיאה: ERROR 6.

ישנן כמובן הודעות שגיאה נוספות שעשויות להתקבל אגב ביצוע הפקודה **ASM** ואנו נעסוק בחלקן בהמשך.

הפקודה BUG

הפקודה **BUG** מאפשרת לך להריץ את התוכנה אשר כתבת באסמבלי ואשר תורגמה לשפת המכונה, ולבדוק אם היא אכן עומדת בציפיות התכנותיות שלך.

זכור כי על מנת לאפשר הרצה של התוכנה, חייבים לסיים בהצלחה את שלב ה-**ASM** – כלומר לקבל דיווח על אפס טעויות בתרגום שורות הקוד לשפת מכונה.

לאחר שתקליד את הפקודה **BUG** האסמבלר ייכנס למצב DEBUG וימתין לפקודת ההרצה.

פקודת ההרצה נפתחת באות **G**, ואחריה ציון כתובת הזיכרון הראשונה שבה שמורה התוכנית בשפת המכונה.

לדוגמה, הפקודה **G600** תריץ את התוכנה השמורה בתא זיכרון 1536 (או $600 בבסיס הספירה ההקסה-דצימלי).

על מנת לצאת ממצב ההרצה ולחזור למצב הכתבן, הקלד את האות **X** והקש ENTER

שים לב – ישנן פקודות נוספות הרלוונטיות למצב ה-DEBUG, ונתייחס לחלקן בהמשך.

הפקודה SAVE

הפקודה **SAVE** משמשת כדי לשמור על גבי דיסקט תוכנה הכתובה בשפת מכונה. שים לב, השימוש בפקודה **SAVE** **לא ישמור את שורות הקוד** שכתבת באסמבלי, אלא **רק את מקטע הזיכרון** שבו שמורה התוכנה המקבילה בשפת מכונה. זאת ועוד, עליך לציין את גבולות מקטע הזיכרון (בשיטה ההקסה-דצימלית) המיועד לשמירה.

אם תרצה למשל לשמור את מקטע הזיכרון מכתובת 1536 עד כתובת 1568 הקלד:

```
SAVE #D:FileName<600,620
```

הפקודה LOAD

הפקודה **LOAD** משמשת כדי לטעון מן הדיסקט תוכנה השמורה בשפת מכונה. שים לב, השימוש בפקודה **LOAD** אינו מאפשר לטעון את שורות הקוד באסמבלי.

צורת כתיבת הפקודה:

```
LOAD #D:FileName
```

עד כאן בנושא הפקודות של כתבן האסמבלר. עתה הגיע הזמן ללמוד את יסודות האסמבלי. בכך נעסוק בפרקים הבאים.

פרק ב
על שלושה רגיסטרים העולם עומד – Y ,X ,A

על מנת לאחוז דבר מה (את העיפרון למשל) תשתמש על פי רוב בידך החזקה, לפעמים בידך השנייה, ולפעמים בשתיהן גם יחד.

האם היית יכול להסתייע ביד שלישית נוספת? כנראה שכן...

באמסמבלי, בכל אופן, קיימות שלוש "ידיים", המכונות רגיסטרים. הרגיסטר המרכזי נקרא "אוגר" (Accumulator) או בקיצור: A, ושני הרגיסטרים הנוספים מכונים: X, Y.

הרגיסטר הראשון שנדון בו הוא רגיסטר A (האוגר). רגיסטר A יכול לבצע את הפעולות הכלליות ביותר מבין "ידי" המחשב, והוא גם עוסק ברוב הפונקציות המתמטיות של המעבד.

ברוב המקרים, על מנת שהמעבד יוכל לבצע פעולה באמצעות רגיסטר A, על הרגיסטר להכיל איזושהי אינפורמציה.

כיצד נגרום לרגיסטר A להכיל דבר מה?

באמצעות הפקודה: **LDA** (שהיא קיצור המילים: LoaD Accumulator).

נתבונן למשל על הפקודה: **LDA $253**

פקודה זו לוקחת את **תוכנו** של תא הזיכרון שכתובתו בשיטה ההקסה-דצימלית: $253 (או: 595 בשיטה העשרונית) ושמה אותו בתוך רגיסטר A (האוגר).

שים לב, על מנת להימנע מטעויות, נעשה לנו מנהג (הן בתכנות, והן ביחידת לימוד זו) להשתמש תמיד בסימן ה-$ לפני כל ערך מספרי הכתוב בבסיס ההקסה-דצימלי.

בתכנות בשפת האסמבלי נשתמש כמעט תמיד בערכים הקסה-דצימליים. הרחבה על שיטת הספירה ההקסה-דצימלית תוכל למצוא בנספח.

הפקודה LDA מאפשרת לאחוז באוגר לא רק ערך שנקרא **מתוך** תא זיכרון, אלא גם ערך בפני עצמו.

נתבונן למשל על הפקודה: LDA #$2E

פקודה זו שמה ברגיסטר A את הערך ההקסה-דצימלי $2E (השווה למספר העשרוני 46).

שים לב: השימוש בסימן הסולמית (#) מציין כי האוגר אוחז בערך עצמו.

התבונן בשתי הפקודות:

```
10      LDA  $18
20      LDA  #$18
```

■ השלם:

הפקודה בשורה ______ תציב באוגר את הערך: $18 ואילו הפקודה בשורה ______ תציב באוגר את הנתון השמור בתא הזיכרון $18.

[תשובה 1 בעמוד 137]

? האם לדעתך תוכל לפקוד על האוגר לאחוז את הערך $1F4 (או 500 בשיטה העשרונית)?

התשובה היא לא!

האוגר יכול לאחוז בכל רגע נתון אך ורק ערכים בין 0 ל-255 (או בין $00 ל-$FF)

אם תפקוד על האוגר לאחוז ערך החורג מן הטווח הזה, כגון:

```
LDA #$1F4
```

כתבן האסמבלר יגיב בהודעת השגיאה: ERROR 10

עד כאן ראינו כיצד לפקוד על האוגר לאחוז ערך מספרי. על מנת לבצע פעולה באמצעות אותו ערך, נוכל להורות למעבד לשים את תוכנו של האוגר בתוך אחד מתאי הזיכרון.

לשם כך נשתמש בפקודה: STA (קיצור המילים: STore Accumulator).

נתבונן למשל על הפקודה: STA $2C8

פקודה זו לוקחת את המידע השמור באוגר ושמה אותו בתוך תא זיכרון $2C8 (או 712 בשיטה העשרונית).

? האם לדעתך קיימת גם פקודה כזו: STA #$2C8 ?

ודאי שלא.

אין שום משמעות להצבת ערך מספרי בתוך ערך מספרי אחר...

בשלב זה, אתה כמעט מוכן לכתוב את התוכנית הראשונה שלך באסמבלי. אך לפני שתעשה זאת, עליך להנחות את כתבן האסמבלר היכן בזיכרון עליו לאחסן את התוכנית המקבילה שהוא ייצר בשפת המכונה.

אחד האזורים בזיכרון שנהוג לכתוב בהם תוכנות בשפת מכונה הוא הטווח שבין בית 1536 לבית 1791 (כלומר הטווח $600-$6FF). אזור זה מכונה גם: "דף 6", משום שאם נתייחס לכל 256 בתים כאל "דף" זיכרון, הרי שהכתובת 1536 מצביעה על הבית הראשון הפותח בפעם השישית כפולה שלמה של 256 בתים.

על מנת להורות לכתבן האסמבלר לייצר את התוכנה בשפת מכונה בדף 6, נפקוד:

```
10    *=$600
```

שים לב, זו תהיה השורה הראשונה שתפתח מעתה ואילך כל תוכנית שנכתוב בשפת האסמבלי.
אם תשכחנה, תדבק לשונך לחכך... וגם תקבל את הודעת השגיאה: ERROR 19

ועוד עניין לפני שתיגש לכתיבת התוכנית הראשונה שלך:
כתבן האסמבלר מצפה לשלושה סימני רווח בין מספר השורה לבין הפקודה. כדי להימנע מתקלות, הקפד על כך.

■ משימה: כתוב תוכנת אסמבלי קצרה שתצבע את המסך בשחור.

רמז: הבית בזיכרון האחראי על צבעו של המסך הוא 710.

[תשובה 2 בעמוד 137]

אל תשכח לפקוד על המחשב: **ASM**, ורק לאחר שווידאת שהמחשב מדווח על אפס טעויות, עבור למצב ההרצה באמצעות הפקודה: **BUG**.

הרץ את התוכנה (שבשפת המכונה) באמצעות הפקודה: **G600**

שים לב, לאחר הרצת התוכנה מופיעה השורה המסכמת הבאה:

S=00 P=30 Y=00 X=00 A=00 0605

■ עתה שנה את התוכנה כך שהאוגר יאחז את הערך $22

אם פעלת נכונה, צבע המסך הפך לאדום והשורה המסכמת התעדכנה:

S=00 P=30 Y=00 X=00 A=22 0605

דע לך כי שורת הסיכום היא כלי עזר חיוני לבדיקת שורות הקוד שכתבת.

כל עוד אתה נמצא במצב ההרצה (המכונה גם: Debugger), תוכל לפקוד על המחשב להציג שוב את השורה המסכמת, על ידי הקלדת הפקודה: **DR**

בשורה המסכמת תוכל לבחון, בין היתר, את מצב הרגיסטרים (A, X, Y) **לאחר** הרצת התוכנה. בנוסף, היא מיידעת אותך כמה בתים בזיכרון תפסה התוכנה שנכתבה בשפת המכונה.

בדוגמה שלפנינו, השורה המסכמת פותחת במספר 605. מספר זה מציין את כתובת הזיכרון האחרונה (כמובן, בשיטה ההקסה-דצימלית) שמקודד האסמבלר השתמש בה, כשקודד את התוכנה שכתבת לתוכנה בשפת מכונה.

מכיוון שהפקודה הראשונה בשפת המכונה נכתבה בכתובת $600, הרי שאורכה של התוכנה כולה הוא 6 בתים בדיוק.

שים לב, אם תרצה לשמור את התוכנה המקודדת בשפת המכונה, תוכל להיעזר במידע על אודות טווח הבתים בהם שמורה התוכנה ולהקליד:

SAVE #D:FileName<600,605

■ נסה!

לאחר ששמרת את התוכנה שלך, כבה את המחשב והפעל אותו מחדש.
עתה טען את התוכנה מן הדיסקט באמצעות הפקודה: **LOAD**
פקוד על המחשב: **LIST**

המחשב מודיע כי לא שמורה בזיכרונו אף שורה!

עתה פקוד על המחשב: **BUG** ומיד אחר כך: **G600**
המסך נצבע באדום.

? מה קרה כאן?

כפי שאמרנו קודם – הפקודה **SAVE** שמרה רק את התוכנה בשפת מכונה אך לא את שורות הקוד באסמבלי.

על מנת לשמור את שורות הקוד באסמבלי, השתמש בפקודה:

LIST #D:FileName.ASM

לפני שנעבור לפרק הבא, אציג לך עוד פקודה שימושית השייכת למצב ההרצה (Debugger).

אם תרצה להחזיר את צבעי המסך לקדמותם, במקום ללחוץ על RESET, תוכל לפקוד: **C 2C6<94**

ובאופן כללי: הפקודה **C** מאפשרת לך – כל עוד אתה במצב ה-Debugger – לפקוד ערך לתוך בית בודד בזיכרון.
צורת השימוש בפקודה הוא:

C בית בזיכרון<ערך

■ עתה צא ממצב ההרצה על ידי הקלדת הפקודה: **X**

פרק ג
שיעור חשבון, הניפו דגל!

האוגר כאמור, הוא היחיד מבין שלושת הרגיסטרים אשר יודע לבצע גם פעולות חשבוניות. אך אל תטפח ציפיות גבוהות מדיי – הפעולות החשבוניות היחידות שהאוגר יודע לבצע הן חיבור וחיסור.*

הפקודה ADC (חיבור עם שארית)

הפקודה האחראית לחיבור מספרים היא ADC שפירושה: חבר עם שארית (ADd with Carry).

על מנת להדגים את אופן פעולתה של הפקודה ניזכר בחיבור במאונך, אשר למדנו בשיעורי החשבון בבית הספר היסודי.

נניח כי אנו מוגבלים לעבודה במספרים דו-ספרתיים בלבד (כלומר עומדת לרשותנו טבלה בת שתי עמודות!), ועלינו לחבר את שני המספרים (העשרוניים) 32 ו-46.

ראשית, נכתוב אותם זה תחת זה, נחבר כל ספרה עם זו שמתחתיה ונקבל מספר דו ספרתי חדש: 78

	3	2
+	4	6
	7	8

* בהמשך נראה שהאוגר יודע לבצע גם גרסה מצומצמת ומוגבלת של פעולות כפל וחילוק.

עד כאן טוב ויפה, אך מה אם נרצה לחבר את שני המספרים: 82 ו-46?
אציג שוב את התרגיל במאונך:

1	8	2
+	4	6
	2	8

הפעם תוצאת החיבור חורגת מן התחום המותר לנו (תחום הדו-ספרתיים), שכן החיבור של ספרת העשרות 8 עם ספרת העשרות 4, מחייב הוספה של ספרה נוספת – ספרת מאות!

האם יש אפשרות לפתור את התרגיל ולהישאר בתחום הדו-ספרתיים?

כן! אם נעצור את פעולת החישוב בדיוק ברגע הזה, **ונזכור** שהנחנו בצד מאה יחידות.

במילים אחרות, נוכל לכתוב כי 82+46 = 28 (+זכירה של 1)

שים לב, זהו בדיוק האופן שבו הורגלנו לעבוד בבית הספר היסודי כשפתרנו תרגילי "חיבור במאונך": במקרה הצורך רושמים בצד (כמו באיור) את ספרת המאות, במטרה להשתמש בה בהמשך פתרון התרגיל.

בדיוק לפי עיקרון זה עובדת פעולת החיבור: **ADC** – אם כי כמובן בשיטת הספירה ההקסה-דצימלית!

כך לדוגמה, אם האוגר נושא את הערך $80 (קרי: 128) ואנו מבקשים להגדילו ב-$40 (קרי: ב-64), נפקוד: **ADC #$40**

תוצאת החיבור תהיה: 128+64=192. תוצאה זו נמצאת בטווח הערכים האפשרי של האוגר, ולכן ערכו החדש של האוגר יהיה: $C0

ואולם, אם האוגר נושא את הערך $80 (קרי: 128) ונבקש להגדילו ב-$90 (קרי ב-144), כאשר נפקוד: `ADC #$90`, תהייה תוצאת החיבור: 272=128+144, וזו תוצאה החורגת מן הערך הגבוה ביותר שהאוגר יכול לקבל (שהוא כזכור: 255).

בכמה חורגת התוצאה? ב-16 (או בשיטה ההקסה-דצימלית, ב-$10)

ולכן ערכו החדש של האוגר יהיה עתה: $10 ונזכור את ה-256 אשר הנחנו בצד.

במילים אחרות, נוכל לכתוב כי: $80+$90 = $10 (+זכירה)

כיצד יודע המחשב לזכור את אותה יחידה ש"הנחנו בצד"?



ובכן, בדיוק לשם כך קיים בתוך המעבד מעין דגלון הנקרא: "דגל השארית" או באנגלית: Carry flag.

כאשר תרגיל החיבור דורש זכירה (כלומר כאשר תוצאת החיבור גדולה מ-255) – דגל השארית (המכונה בקיצור: C) "נדלק" ומקבל את הערך 1. במילים אחרות, C משמש למעשה **כעדות** – עדות לכך שתרגיל החיבור שביצענו קודם לכן דרש זכירה.

שים לב, הדגל C אינו נכבה אם לא מכבים אותו. זאת ועוד – כל עוד הדגל C דולק, כל פעולת חיבור נוספת שנפעיל על האוגר, תגדיל את תוצאת החיבור ב-1.

כך לדוגמה, אם דגל C דולק, תוצאת התרגיל 5+3 תהייה 9 (ולא 8!)

? למה הדבר מועיל?

לפני שנשיב על השאלה, נעשה אתנחתא לרגע, וניזכר בשיטת "הבתים המצביעים" אשר פגשנו לראשונה במחשבת 6.

ניקח לדוגמה את זוג הבתים המצביעים: 88, 89 אשר כזכור נושאים יחדיו את כתובת התא הראשון של זיכרון המסך. במסך גרפי 0 למשל, הערך השמור בבית 89 הוא: 156, ואילו הערך השמור בבית 88 הוא: 64, ויחדיו הם מצביעים על תא זיכרון 40,000.

עד כה – כשעבדנו בסביבת הבייסיק – נהגנו לחשב באופן די מייגע את הכתובות הללו באופן הזה:

156 X 256 + 64 = 40,000

בשיטה ההקסה-דצימלית, לעומת זאת, כל שנדרש לעשות הוא להצמיד את שני הערכים זה לצד זה. כך, אם ערכו של בית 89 הוא $9C, וערכו של בית 88 הוא $40, אם נצמיד את הערכים זה לזה נקבל: $9C40 (שהם 40,000 בשיטה העשרונית).

עכשיו, נניח שאנו רוצים להקפיץ את זיכרון המסך קדימה ב-20 יחידות. כל שעלינו לעשות הוא להגדיל את הערך $40 ל-$54 ולפקוד ערך זה לתוך בית 88. כך נקבל את הכתובת החדשה: $9C54

ומה אם נרצה להקפיץ את זיכרון המסך קדימה ב-200 יחידות? הפעם, אם נגדיל את הערך $40 ב-200 יחידות נקבל $108 (שהם 264), אך לא נוכל לפקוד ערך שכזה לתוך בית 88, שהרי הערך המקסימלי שבית יכול לשאת הוא $FF!

במקום זאת, נצטרך "לשים בצד" 256 יחידות, לפקוד את היתרה – 8 יחידות – לתוך בית 88. ובתמורה, לעבור לבית השני, בית 89, ולהגדילו ביחידה אחת מ-$9C ל-$9D.

כך נקבל את כתובת הזיכרון החדשה: $9D08 (או 40,200 בשיטה העשרונית).

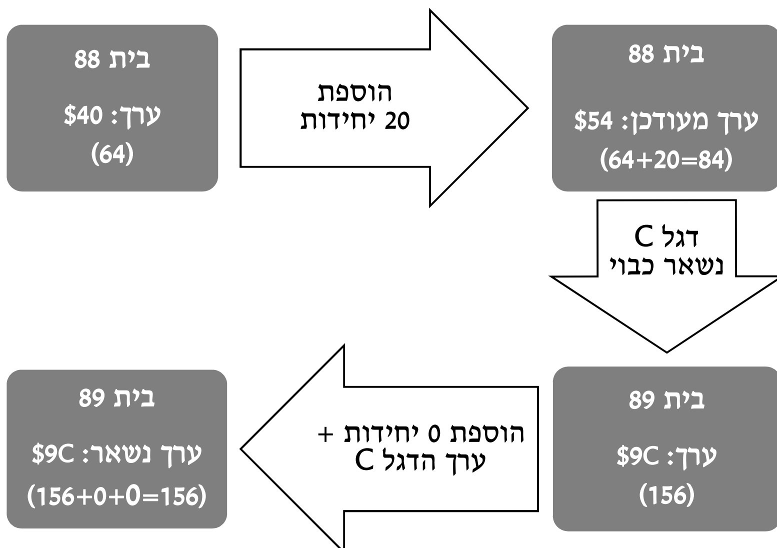
נחזור עתה לנושא שבו פתחנו ונדגים כיצד דגל השארית (C) יכול להיות לנו כאן לעזר.

על מנת להקפיץ קדימה את זיכרון המסך (כל קפיצה, בצעדים של בין 1 ל-255 יחידות), כל שעלינו לעשות הוא לבצע – באמצעות האוגר כמובן – **תמיד שתי פעולות חיבור בזו אחר זו**: בפעולת החיבור הראשונה להוסיף את טווח הקפיצה לערכו של בית 88 ובפעולת החיבור השנייה להוסיף 0 לערכו של בית 89.

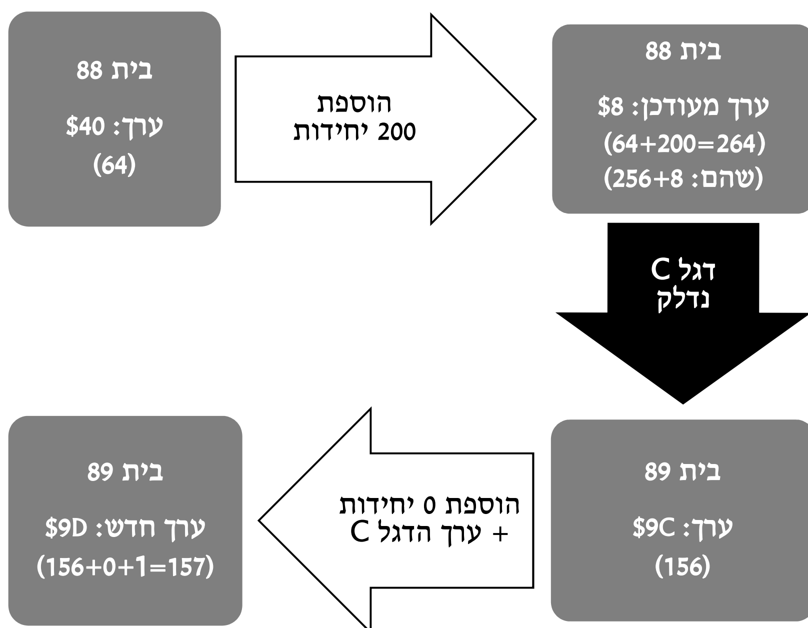
אם פעולת החיבור הראשונה גרמה להדלקת דגל השארית, הרי שפעולת החיבור השנייה תגדיל את בית 89 ב-1. אך אם פעולת החיבור הראשונה לא גרמה להדלקת דגל השארית, הרי שערכו של בית 89 לא ישתנה.

נמחיש זאת בעזרת שני התרשימים הבאים.

מקרה ראשון:



מקרה שני:



זכור כי **איפוס הדגל** C **הוא באחריות המתכנת** – כלומר באחריותך!

על כן, עשה לך מנהג **לכבות תמיד את הדגל** C **לפני כל פעולת חיבור** חדשה שתפעיל על האוגר, אלא אם ברצונך להיעזר במצב הדגל לפעולות החיבור הבאה (כפי שראינו בדוגמה לעיל).

וכאן המקום להציג את פקודת כיבוי דגל השארית, הפקודה: **CLC** (שהיא קיצור המילים: CLear Carry).

■ כתוב תוכנה קצרה באסמבלי שתצבע את המסך ואת המסגרת שלו בשני הגוונים המנוגדים – החום כהה ביותר (המקודד בערך העשרוני: 16+1) מול הצהוב הבהיר ביותר (המקודד בערך העשרוני: 16+15). לצורך התרגול, עליך להשתמש פעם אחת ויחידה בפקודה LDA!

אל תשכח לכבות את דגל השארית לפני ביצוע פעולת החיבור.

[תשובה 3 בעמוד 137]

הרץ את התוכנה שכתבת ושים לב לשורה המסכמת המופיעה בסיום הרצת התוכנה. האוגר (A) אוחז בערך $1F (כלומר: 31) בדיוק כפי שציפינו, לאחר שבוצעה פעולת החיבור.

■ הוסף כעת את שתי השורות הבאות:

```
12      LDA  #$96
14      CLC
18      ADC  #$96
```

כמו כן, מחק את שורה מספר 40.

השורות שהוספנו לתוכנה פוקדות על האוגר לבצע משימה מקדימה של חיבור שני מספרים.

? האם לדעתך תהייה לכך השפעה על המשימה השנייה של צביעת המסך?

■ בדוק!

שים לב לשינוי שקרה: מסגרת המסך שינתה את צבעה, ובשורת הסיכום ניתן לראות כי האוגר נושא בערך גבוה יותר ממה שציפינו: $20 (כלומר: 32).

ההסבר לכך הוא פשוט. פעולת החיבור הראשונה ($96+$96) הכניסה לאוגר ערך גבוה מן התחום המותר וגרמה להדלקת דגל השארית (C). כתוצאה מכך, פעולת החיבור הבאה הגדילה את התוצאה ביחידה אחת.
למעשה, עד אשר נפקוד על המחשב: **CLC**, כל פעולת חיבור נוספת שיבצע האוגר תיתן תוצאה הגדולה מסכום המספרים ביחידה אחת.

הפקודה SBC (חיסור עם שארית)

הפקודה האחראית לחיסור מספרים היא **SBC** שפירושה: חסר עם שארית (SuBtract with Carry).
פעולת החיסור פועלת בצורה הפוכה מפעולת החיבור. על מנת להדגים את עקרון עבודתה נעבור לרגע לתחום העשרוני, ונדמיין כי האוגר הוא קונה אשר נדרש **לפעמים** לגשת לבנק ולקחת הלוואה.
אם לרשות האוגר 46 שקלים והוא מבקש לרכוש מוצר ב-32 שקלים, אין כל בעיה. נחסר 32 מ-46 ונקבל: 14

	4	6
-	3	2
	1	4

אך מה עם עומדים לרשות האוגר 46 שקלים והוא מבקש לרכוש מוצר ב-52 שקלים? כאן נדרשת הלוואה. "הבנק" מלווה לאוגר מאה שקלים, ומגדיל בכך את "הונו" של האוגר ל-146. עתה יכול האוגר לשלם את הסכום. בידיו של האוגר נותרים 94 שקלים וגם זיכרון (כואב) שהוא לוֶוה בעל חוב.

אפשר לרשום זאת גם כך:

1	4	6
-	5	2
	9	4

לאחר שהעיקרון הובהר, אציג את פקודת החיסור: **SBC**, אשר עובדת, כמובן, בשיטת הספירה ההקסה-דצימלית.

השימוש בפקודת החיסור **SBC** מניח שהאוגר **עשוי** להזדקק להלוואה. לכן <u>לפני</u> פעולת החיסור נרצה "לשמור בצד" את אותה הלוואה, וכך נוכל לבדוק אם זו אכן נלקחה.

לשם כך, לפני פעולת החיסור נפקוד: **SEC**. פקודה זו (שהיא קיצור המילים: SEt Carry), תדליק את דגל השארית (C) ותקבע אותו על 1.

נבחן שתי דוגמאות:

1. אם האוגר נושא את הערך $40 (קרי: 64) ואנו מבקשים להקטינו ב-$30 (קרי: ב-48), נפקוד: **SBC #$30**

תוצאת החיבור תהיה: 16=64-48. פעולה זו לא דרשה הלוואה, ולכן ערכו החדש של האוגר יהיה: $10 וערכו של דגל השארית יישאר: 1.

2. אם האוגר נושא את הערך $40 (קרי: 64) ונבקש להקטינו ב-$50 (קרי ב-80), כאשר נפקוד: **SBC #$80** האוגר יהיה חייב "לקחת את ההלוואה" של $100 (קרי: 256), ולכן תוצאת החיסור תהייה: 240=80-(64+256) – כלומר $F0 ודגל השארית ייכבה וייקבע על 0.

במילים אחרות, בכל הנוגע לפעולות החיסור, דגל השארית C משמש למעשה **כעדות** – עדות לכך שנלקחה (או לא נלקחה) הלוואה.

שים לב, הדגל C אינו נדלק מחדש אם לא מדליקים אותו. זאת ועוד – כל פעולת חיסור שנפעיל על האוגר, תפחית מן התוצאה את **ערך היפוכו** של הדגל.

כך לדוגמה, אם הדגל C דולק (ערכו: 1), תוצאת התרגיל 8-3 תהייה 5

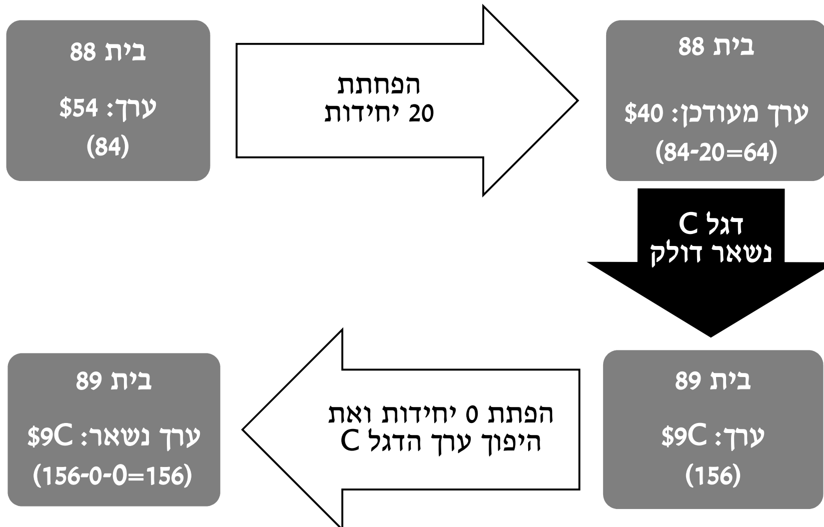
אך אם הדגל C מכובה (ערכו: 0), תוצאת התרגיל 8-3 תהייה 4!

גם הפעם נדגים את השימוש בדגל השארית בפעולות חיסור באמצעות זוג הבתים המצביעים: 88 ו-89. על מנת להקפיץ **אחורה** את זיכרון המסך, כל שעלינו לעשות הוא לבצע באמצעות האוגר **תמיד שתי פעולות חיסור** בזו אחר זו: בפעולת החיסור הראשונה להפחית את טווח הקפיצה מערכו של בית 88 ובפעולת החיסור השנייה להפחית 0 מערכו של בית 89.

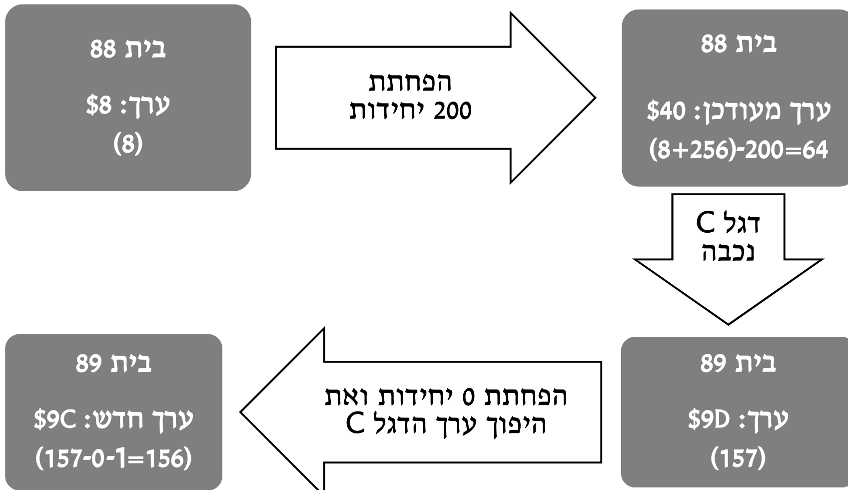
אם פעולת החיסור הראשונה גרמה לכיבוי דגל השארית, הרי שפעולת החיסור השנייה תקטין את בית 89 ב-1 (שהוא היפוכו של דגל כבוי). אך אם פעולת החיסור הראשונה לא גרמה לכיבוי דגל השארית, הרי שערכו של בית 89 לא ישתנה.

נמחיש זאת שוב בעזרת שני תרשימי זרימה המתארים את החזרת מיקום זיכרון המסך למצבו ההתחלתי.

מקרה ראשון:



מקרה שני:



זכור כי **הדלקת הדגל C היא באחריות המתכנת** – כלומר באחריותך!

על כן, עשה לך מנהג **להדליק תמיד את הדגל** C, **לפני כל פעולת חיסור** חדשה שתפעיל על האוגר, אלא אם ברצונך להיעזר במצב הדגל לפעולות החיסור הנוספות (כפי שראינו בדוגמה לעיל).

■ כתוב תוכנה קצרה באסמבלי שתכתוב בפינה העליונה השמאלית של המסך את המילה: on. לצורך התרגול, עליך להשתמש פעם אחת ויחידה בפקודה **LDA**.

רמז: את ערכי ה-ASCI של התווים הרצויים תוכל לפקוד ישירות למקום המתאים בזיכרון המסך.

ושוב, נזכיר לך להדליק את דגל השארית (C) לפני ביצוע פעולת החיסור.

[תשובה 4 בעמוד 137]

שים לב, כדי שהמילה "on" לא תיעלם כהרף עין מן המסך, לחץ על RESET, פקוד שוב: **BUG** והרץ את התוכנה על מסך נקי יותר.

■ עתה מחק את שורה מספר 40 והקלד במקומה:

```
40   CLC
```

? לפני שתריץ את התוכנה חשוב מה יקרה לאחר איפוס דגל השארית: האם תופיע המילה on, oo או om?

■ בדוק האם צדקת!

סיכום הפרק

בפרק ב' הכרנו את פקודת החיבור: **ADC**, ואת פקודת החיסור: **SBC**.

זכור כי פעולות החיבור והחיסור הן ייחודיות לרגיסטר A – האוגר.

כמו כן הכרנו את דגל השארית C המאפשר לבצע תרגילי חיבור וחיסור במספרים גדולים או קטנים יותר מגבול יכולתו של האוגר.

הפקודה **CLC** מכבה (מאפסת) את דגל השארית. כאשר דגל השארית דולק, כל תרגיל חיבור יגדיל את התוצאה ביחידה אחת.

הפקודה **SEC** מדליקה את דגל השארית (קובעת את ערכו כ-1). כאשר דגל השארית כבוי, כל תרגיל חיסור יקטין את התוצאה ביחידה אחת.

לבסוף, ראינו את היתרון העצום בשימוש בשיטה ההקסה-דצימלית בכל הנוגע לעיסוק בבתים-המצביעים – יתרון המתבטא כמובן במהירות הפנייה לתאי הזיכרון השונים.

פרק ד
הולך וחוזר חלילה

אחד המנגנונים היסודיים ביותר בכל שפת תכנות היא הלולאה המותנית. כלומר, היכולת לחזור על סדרה של פעולות שוב ושוב עד אשר מתקיים תנאי מסוים.

הפקודה האחראית ליצירת לולאות בשפת האסמבלי היא **JMP** (קיצור המילה JuMP) ועקרון פעולתה דומה מאוד לעקרון פעולתה של פקודת הבייסיק: **GOTO** – המחשב מקבל הוראה לקפוץ בחזרה לנקודה מסוימת, לבצע פעולה או פעולות מסוימות וחוזר חלילה.

ואולם, על מנת להורות למחשב לאיזו נקודה עליו לחזור, חייבים להגדיר את אותה נקודה. הדרך היעילה ביותר לעשות זאת היא באמצעות תווית, כלומר רצף כלשהו של כמה אותיות (רצוי בעל משמעות). את התווית הזו נניח בדיוק בנקודה שאליה נרצה שהמחשב יחזור כשנפקוד עליו **JMP**.

על מנת להדגים את אופן פעולתה של הלולאה ניזכר בשעון הפנימי של המחשב השמור בבתים: 18, 19, 20 (אם אינך זוכר, עיין בפרק ב׳ של מחשבת 6).

נבנה תוכנית קצרה שקוראת (באמצעות האוגר) את ערכו של תא הזיכרון ‎$14 (קרי: 20) האחראי על ה"שניות" ופוקדת את הערך הזה לתוך הבית ‎$2C8 (קרי: 710) – אשר אחראי כזכור, לצבע מסגרת המסך.

על מנת ליצור לולאה נגדיר תווית בשם: LOOP ונפקוד על המחשב לדלג אליה.

```
10     *=$600
20 LOOP LDA $14
50     STA $2C8
60     JMP LOOP
```

שים לב, כאשר אתה מקליד פקודה באסמבלי, כתבן האסמבלר מצפה כזכור לשלושה סימני רווח בין מספר השורה לבין הפקודה. לעומת זאת, כאשר אתה מקליד תווית (כמו בשורה 20) **הקפד לשמור על רווח אחד בלבד** בין מספר השורה לבין התווית. אחרת התוכנית לא תפעל כנדרש!

■ הרץ את התוכנה (לאחר שהקלדת כמובן: **ASM**), אך לפני שתעשה זאת זכור כי הלולאה היא אינסופית ובלתי ניתנת לעצירה – גם לא על ידי הקשה על מקש: BREAK. הדרך היחידה שבה תוכל לעצור את התוכנה היא באמצעות לחיצה על מקש: RESET.

התבונן בצבעים המלהיבים המתחלפים בקצב מסחרר, והקש על מקש: RESET.

לפני שנעבור לתת הסעיף הבא, נדגיש כמה עניינים לגבי אותן תוויות:

1. התוויות **אינן חלק משפת המכונה,** אלא הן משמשות אך ורק ככלי עזר לצורך התכנות באסמבלי. כאשר מקודד האסמבלר פוגש בתווית שיצרת, הוא מחשב את כתובת הזיכרון הספציפית שאליה אמור המעבד לפנות, ומתרגם את התווית לכתובת המספרית הזו.

2. אין מניעה להשתמש בכל רצף של אותיות (או אותיות בצירוף ספרות), ובתנאי שרצפים אלו אינם משמשים פקודות אסמבלי. כך למשל, אין להגדיר תווית בשם: LDA. וגם לא תוויות בשם: X, Y, או A (שמותיהם של הרגיסטרים).

3. כאשר אתה משתמש בתוויות, הקפד להקליד נכונה את אותיות התווית. אם מקודד האסמבלר יפגוש התייחסות לתווית שלא הוגדרה מבעוד מועד, תתקבל הודעת השגיאה: **5** ERROR

אם תגדיר שתי תוויות באותו שם, תתקבל הודעת השגיאה: **7** ERROR

לולאות מותנות ודגל האפס

על מנת שנוכל לעצור את הלולאה (מבלי שנזדקק להקיש על מקש: RESET), נדרשים שני דברים: ראשית פקודת התניה, אשר יודעת לבחון אם מתקיים תנאי לוגי מסוים. ושנית פקודת דילוג, המאפשרת לקפוץ החוצה מן הלולאה אם וכשאר מתקיים אותו תנאי.

אחת הפקודות המאפשרות לנו לבנות את אותה התניה היא הפקודה: **CMP** (קיצור המילה: CoMPare). פקודה זו משווה בין שני ערכים – בין הערך השמור **באוגר** לבין ערך מספרי כלשהו.

ככלל, כאשר אנו משווים בין שני ערכים יכולים להתקיים אחד משני מצבים: או שערכו של א' שווה לערכו של ב'; או שערכו של א' שונה מערכו של ב'. בנוסף, נוכל לנסח שני מצבים נוספים הקשורים באותה השוואה: או שערכו של א' גדול מערכו של ב'; או שערכו של א' קטן מערכו של ב'.

באסמבלי ישנם אפוא ארבעה מבנים של פקודות תנאי – כנגד כל אחד מסימני השוויון (=, ≠, >, <) – כולם בעלי עקרון פעולה דומה. נעבור על כל אחד מהם בנפרד.

(1) זוג הפקודות CMP ו-BEQ



הפקודה **CMP** יודעת כאמור להשוות בין שני ערכים, ולצורך כך היא משתמשת בדגל האפס (Zero flag) או בקיצור: Z.

אם מתקיים שוויון בין הערכים, דגל האפס (Z) נדלק (מקבל את הערך 1). אם לא מתקיים שוויון בין הערכים, דגל האפס (Z) נכבה (מקבל את הערך 0).

? וכיצד נעזרים במצבו של הדגל?

באמצעות הפקודה **BEQ** (קיצור המילים: Branch EQual)

הפקודה **BEQ** מבצעת דילוג (קפיצה החוצה מתוך הלולאה) כאשר ערכו של הדגל Z שווה ל-1 (כלומר כשמתקיים שוויון בין ערכו של האוגר לערך מסוים).

כמובן שעלינו להודיע למחשב להיכן לקפוץ כאשר מתממש תנאי השוויון, ולצורך כך נשתמש בתווית נוספת.

כדי להדגים את צורת העבודה של תנאי השוויון, נחזור לתוכנית שכתבנו בסעיף הקודם (צבעי הרקע המתחלפים לפי קצב "שניות" המחשב) ונוסיף את פקודת ההשוואה והקפיצה מן הלולאה.

■ הוסף את שלוש השורות הבאות לתוכנה:

```
30      CMP  #$FF
40      BEQ  EXIT
70  EXIT
```

(הקפד להקליד רווח אחד בלבד בשורה 70 בין מספר השורה לבין התווית)

הרץ את התוכנה (לאחר שפקדת על המחשב: **ASM**) וראה את התוצאה. חזור כמה פעמים על הפקודה: **G600**

מה מתרחש פה בעצם?

המחשב טוען שוב ושוב את ערך "שניות המחשב" לאוגר ובודק אם הערך שווה ל-255. כאשר מתקיים השוויון מתבצעת קפיצה החוצה מן הלולאה וריצוד הצבעים נפסק.

? אגב, האם ניכר הבדל בין הרצה אחת של התוכנה להרצה אחרת?

בוודאי שכן. מכיוון ששעון המחשב ממשיך כל העת לרוץ (גם כשהתוכנה שלך מסתיימת), לפעמים ייקח זמן רב עד ש"מחוג השניות" ישלים את "מסלולו" ויגיע ל-255, ולפעמים זמן קצר יותר...

(2) זוג הפקודות CMP ו-BNE

לפעמים נרצה לעבוד בצורה הפוכה. כלומר לקפוץ החוצה מן הלולאה רק אם לא מתקיים תנאי מסוים.

גם הפעם ניעזר בפקודת ההשוואה **CMP**, אשר קובעת את מצבו של דגל האפס (Z). אך הפעם נצרף אליה את פקודת הדילוג **BNE** (קיצור של Branch Not Equal).

פקודה זו מבצעת קפיצה החוצה מתוך הלולאה כאשר **לא** מתקיים השוויון – כלומר כאשר ערכו של הדגל Z שווה ל-0. גם כאן נשתמש בתווית על מנת להודיע למחשב להיכן לקפוץ.

■ כתוב כעת תוכנה קצרה הבוחנת את מצבו של הג'ויסטיק שלך, אגב סימון תו מתחלף על קצהו השמאלי העליון של המסך.

כל עוד הג'ויסטיק במצב מנוחה (ערך: 15) המחשב ימתין, ורק כאשר תזיז את הג'ויסטיק לאחד הכיוונים, התו ישתנה והתוכנה תסתיים.

רמז: הבית האחראי על תנועת הג'ויסטיק הוא 632 (תוכל גם לעיין בפרק ג' במחשבת 7 – גרפיקת השחקנים).

[תשובה 5 בעמוד 138]

■ לפני שנעבור לשני המבנים האחרונים של התניות באסמבלי (המטפלים ביחסים מסוג <, >), מחק את התוכנה שכתבת והקלד במקומה את התוכנה הקצרה הבאה, המבצעת פעולת חיסור:

```
10    *=$600
20    LDA  #$8
30    SEC
50    SBC  #$3
```

הרץ ובדוק את השורה המסכמת בתחתית.

ערכו של האוגר (A) הוא כמובן: 5 (תוצאת ההפחתה של 3 מ-8)

■ הוסף עתה את השורה הבאה:

```
40   CMP  #$2
```

הרץ את התוכנה (אל תשכח לפקוד: ASM) ובדוק שוב את שורת המצב בתחתית.

הוספת פקודת ההשוואה לא גרמה לשינוי וערכו של האוגר (A) נותר: 5

■ שנה עתה את שורה 40 והקלד:

```
40    CMP  #$9
```

הרץ ובדוק שוב...

שים לב! ערכו של האוגר השתנה והוא שווה עתה ל-4 (!)

? מה בעצם קרה כאן?

מסתבר כי הפקודה CMP השפיעה איכשהו על פעולת ההפחתה!

? האם אתה זוכר מצב נוסף בו השפיע דבר מה על פעולת ההפחתה?

בוודאי! בפרק הקודם פגשנו את דגל השארית (C) אשר משפיע על פקודת ההפחתה, במובן זה שאם הוא כבוי (כלומר ערכו 0) תוצאת ההפחתה תקטן ביחידה נוספת אחת.

אז... כפי שאומר הפתגם האמריקני, אם זה נראה כמו ברווז, הולך כמו ברווז ומגעגע כברווז, זה ברווז! כלומר – הפקודה CMP, בנוסף על השפעתה על דגל האפס (Z), משפיעה גם על דגל השארית (C)!

כל זה לא אמור להפתיע אם נבין שפקודת ההשוואה CMP מתנהגת למעשה כפעולת חיסור לכל דבר – היא משווה שני מספרים על ידי חיסורם זה מזה!

אם תוצאת החיסור נותנת מספר חיובי (דהיינו ערך האוגר גדול מן הערך המושווה), ערכו של דגל C לא משתנה והוא נותר 1 (דולק), אך אם תוצאת החיסור נותנת מספר שלילי (דהיינו ערך האוגר קטן מן הערך המושווה), ערכו של דגל C הופך ל-0 (נכבה).

עתה נוכל לנסח את שתי פקודות ההתניה הנוספות:

BCS מטפלת במצב בו ערכו של האוגר גדול מערך מסוים.

BCC מטפלת במצב שבו ערכו של האוגר קטן מערך מסוים.

נפרט:

(3) זוג הפקודות CMP ו-BCS

הפקודה BCS (קיצור המילים: Branch Carry Set) מאפשרת לדלג מתוך לולאה אם דגל C דולק – כלומר אם וכאשר הערך השמור באוגר גדול מערך מסוים.

■ הקלד את התוכנית הבאה:

```
10      *=$600
20      LDA  #$0
30      STA  $2FC
40  LOOP LDA  $2FC
50      STA  $9C40
80      JMP  LOOP
```

הרץ את התוכנית והקש על מקשי המקלדת. שים לב, בנקודה השמאלית העליונה של המסך מופיע תו מסוים, והוא משתנה עם כל מקש שתקיש במקלדת.

דע לך כי בית 764 (או $2FC) מנטר את לוח המקשים של המחשב, ונושא את ערך המקש האחרון שהוקלד. שים לב, הערך המספרי הזה אינו זהה לערך ה-ASCII של התו המוקלד, אלא שלכל מקש במקלדת קבוע ערך ייחודי משלו.

בשורות הקוד 20-30 איפסנו את ערכו של הבית 764, ואילו בשורות 40-70 יצרנו לולאה אינסופית אשר אין שום דרך לצאת ממנה, זולת לחיצה על מקש RESET.

לצורך התרגול של סעיף זה, אציג לך כעת את ערכיו של בית 764 עבור הקשה על מקשי הספרות 0-9. (תוכל כמובן למצוא בעצמך את ערכו של כל מקש ומקש בעזרת תוכנת בייסיק פשוטה).

מקש	0	1	2	3	4	5	6	7	8	9
ערך	50	31	30	26	24	29	27	51	53	48

■ עתה אבקש ממך להוסיף כמה שורות קוד לתוכנה שלעיל, כך שהיא תגיב למקשי הספרות 1, 2, 3, 4, 5 ו-6, אך תעצור אם נלחצו מקשי הספרות: 7, 8, 9, או 0.

[תשובה 6 בעמוד 138]

(4) זוג הפקודות CMP ו-BCC

הפקודה BCC (קיצור המילים: Branch Carry Clear) מאפשרת לדלג מתוך לולאה אם דגל C מכובה – כלומר אם וכאשר הערך השמור באוגר קטן מערך מסוים.

■ שנה את שורות הקוד הנדרשות כדי שהתוכנית הקטנה שכתבנו תגיב עתה רק למקשי הספרות 7, 8, 9, ו-0, אך תעצור אם נלחצו מקשי הספרות: 1, 2, 3, 4, 5 או 6.

רמז: הפעם מוטב לא לאפס את ערכו של בית 764 אלא דווקא להציב בו ערך התחלתי גבוה.

[תשובה 7 בעמוד 138]

סיכום הפרק

בפרק זה למדנו לבנות לולאות אינסופית באמצעות השימוש בפקודה: **JMP**. ראינו כי על מנת להורות למחשב לאן לקפוץ, נוכל להיעזר בתווית – רצף קצר של כמה אותיות, ורצוי בעל משמעות המובנת לך.

הכרנו גם את הפקודה **CMP** המשווה (באמצעות פעולת חיסור) בין ערכו של **האוגר** לערך מספרי כלשהו ולמדנו להיעזר בפקודה זו על מנת לבנות לולאות מותנות.

הפקודות **BEQ** ו-**BNE** משמשות לדילוג מלולאה, אם מתקיים בין שני הערכים המושווים יחס של שוויון או אי-שוויון, בהתאמה.

הפקודות **BCS** ו-**BCC** משמשות לדילוג מלולאה, אם מתקיים בין שני הערכים המושווים יחס של גדול מ- או קטן מ-.

לבסוף, הכרנו דגל חדש – דגל האפס (Z) אשר מגיב אף הוא – בנוסף לדגל השארית (C) – לפקודת ההשוואה **CMP**.

פרק ה
תפעיל לי מונה, בבקשה!

את הפרק הקודם העוסק בלולאות מותנות פתחנו בדוגמה צבעונית, ונעזרנו ב"שניות" המחשב המתחלפות במהירות על מנת לצבוע את רקע המסך בגוונים מתחלפים.

האם קיימת דרך אחרת לגרום למחשב למנות מספרים מ-0 עד 255 (קרי: $FF), מבלי להסתמך על אותו שעון פנימי?

בוודאי!

אך ראשית, נחזור אל שני מכרים ותיקים שהזנחנו מעט לאחרונה, הלוא הם הרגיסטרים X ו-Y.

בפרק ב הכרנו את הפקודה: **LDA**, אשר מורה למחשב לטעון ערך לתוך רגיסטר A; ואת הפקודה: **STA**, אשר מורה למחשב להציב את הערך השמור ברגיסטר A אל תוך תא זיכרון.

דע לך כי גם עבור הרגיסטרים X ו-Y קיימות פקודות מקבילות, הגיעה העת להכירן:

בדומה לפקודה: **LDA**, גם הפקודות **LDX** ו-**LDY** טוענות ערך לתוך רגיסטר.

לדוגמא:

LDX $2B תטען ברגיסטר X את ערך הבית השמור בכתובת הזיכרון $2B (או 43 בספירה העשרונית).

LDY #$F תטען ברגיסטר Y את הערך $F (או 15 בספירה העשרונית).

ובדומה לפקודה: **STA**, הפקודות **STX**, ו-**STY** משמשות להצבת הערך השמור ברגיסטר אל תוך תא זיכרון.

לדוגמא:

$2F4 STX תציב את הערך השמור ברגיסטר X בכתובת הזיכרון 756.

$9C40 STY תציב את הערך השמור ברגיסטר Y בכתובת הזיכרון $9C40 (או 40,000 בספירה העשרונית).

מכיוון שרגיסטר A הוא היחיד, כאמור, ש"יודע חשבון", נייחד אותו למשימות חשבוניות, ונעדיף להשתמש בשני הרגיסטרים X ו-Y לפעולות שאינן חשבוניות.

אחת הפעולות שנעדיף לגייס עבורן את הרגיסטרים X ו-Y היא מנייה. ולצורך כך קיימים שני זוגות של פקודות ייחודיות אך ורק לרגיסטרים הללו.

הפקודות: **INX** (קיצור של: INcrement X) ו-**INY** (קיצור של: INcrement Y) אשר מגדילות את ערכו של רגיסטר X, או רגיסטר Y ביחידה אחת.

והפקודות **DEX** (קיצור של: DEcrement X) ו-**DEY** (קיצור של: DEcrement Y) אשר מקטינות את ערכו של רגיסטר X, או רגיסטר Y ביחידה אחת.

נחזור לדוגמת צבעי רקע המסך המתחלפים במהירות, וננסה לבנות תוכנית קצרה שמחליפה את צבעי הרקע במחזור אחד של התקדמות (מ-0 ל-$FF). הפעם במקום להשתמש ב"שניות המחשב", ניעזר בפקודת המנייה **INX** (או **INY**)

על מנת לעצור את לולאת המנייה (למשל בדיוק כשהמונה מגיע ל-$FF) ניעזר בפקודת ההשוואה – אך הפעם לא בפקודה **CMP**, אלא בפקודה המקבילה: **CPX** עבור רגיסטר X, או: **CPY** עבור רגיסטר Y.

■ הקלד את שורות הקוד הבאות:

```
10    *=$600
20    LDX #$0
30 LOOP INX
40    STX $2C8
50    CPX #$FF
60    BEQ EXIT
70    JMP LOOP
80 EXIT
```

ודא כי אתה מבין כל אחת משורות הקוד שהקלדת, בצע **ASM** והרץ את התוכנה.

? האם התוצאה משביעת רצון?

נראה שלא... במקום לקבל ריצוד מרהיב של צבעים, הרקע נצבע בצהוב והתוכנה הסתיימה.

? האם יש לך מושג מדוע?

ובכן, מסתבר שפגשת כרגע את "השולף הכי מהיר במערב"! קצב תחלופת הצבעים היה כל כך מהיר עד כי העין שלך לא הייתה מסוגלת לקלוט כלל את הריצוד, וכהרף עין הוא הסתיים.

שים לב להבדל בין התוכנה הקודמת שהייתה מבוססת על מנייה באמצעות תחלופת "שניות" המחשב לבין התוכנה הזו, המבוססת על מנייה של המעבד עצמו. אמנם אורכה של "שניית" מחשב נראה לך קצר מאוד (כזכור, בשנייה רגילה נכנסות 50 "שניות מחשב"!), ואולם פעימת המעבד קצרה ממנה עשרות

מונים (רק כדי לסבר את האוזן – בשנייה רגילה מבצע המעבד למעלה מ-1,500 פעולות!).

- מחק כעת את ההתניה שבתוך הלולאה (כלומר את שורות 50, 60), פקוד: **ASM** והרץ את התוכנה מחדש...

הפעם הריצוד (או יותר נכון ההבזק) של הצבעים מהיר ביותר משום שהוא חוזר על עצמו שוב ושוב ללא הפסק. בכל אימת שהרגיסטר מגיע לערכו המקסימלי ($FF) הוא מתאפס ומתחיל לגדול מחדש, וחוזר חלילה, עד אשר תקיש על מקש RESET.

? האם נוכל להאט מעט את הקצב?

בהחלט.
לצורך כך נוכל למשל להיעזר שוב ב"שניות המחשב" אך הפעם לא לצורך תחלופת הצבעים עצמה, אלא לצורך יצירת **לולאת השהייה**.

- הוסף כעת לולאת השהייה אשר תגרום לצבעי רקע המסך להתחלף אחת לשנייה (שנייה רגילה).

רמזים:

(1) תוכל לאפס שוב ושוב, ברגע שתבחר, את בית 20 ($14) המונה את "שניות המחשב".
(2) על מנת ליצור את לולאת ההשהיה, עליך להיעזר ברגיסטר נוסף, כגון: רגיסטר Y.

[תשובה 8 בעמוד 139]

עתה, לאחר שהתנסינו בפקודה **INX** (או **INY**), נוודא שאנו שולטים גם בפקודה הנגדית **DEX** (או **DEY**).

■ כתוב כעת תוכנה דומה אשר מחליפה את צבעי רקע המסך בקצב מהיר של החלפה אחת בכל עשירית השנייה. הפעם התחל מערך הצבע המקסימלי ($FF) והתקדם מטה עד לערך הצבע המינימלי ($0).

לצורך התרגול, החלף הפעם בין הרגיסטרים X ו-Y

[תשובה 9 בעמוד 139]

התוכנה שכתבת אינה ניתנת לעצירה (אלא בהקשה על מקש: RESET).
אם תרצה להוסיף לה פקודת עצירה (כלומר דילוג מתוך הלולאה המרכזית) תוכל כמובן להוסיף עוד פקודת השוואה (**CPY**) ואחריה פקודת **BEQ** שתדלג לתווית חדשה בקצה התוכנית.

ואולם יש דרך אלגנטית יותר לעשות זאת.

שים לב – הפקודות **DEY**, ו-**INY** (וכמובן גם **DEX** ו-**INX**) משפיעות כשלעצמן הן על דגל האפס (Z) והן על דגל השארית (C).
ולכן, אם תרצה למשל שהמחשב ידלג החוצה מן הלולאה בדיוק ברגע שבו רגיסטר Y הגיע ל-0, תוכל להיעזר במצבו של דגל האפס Z, ולהוסיף פקודת דילוג מתאימה.

■ הוסף פקודת דילוג שתעצור את הלולאה אחרי מחזור שלם של מעברי צבעים (כלומר כשערך הצבע קטן ומגיע ל-0). אל תשכח להוסיף תווית שתציין להיכן על המחשב לדלג בסיום.

[תשובה 10 בעמוד 140]

מתקדמים בעזרת המונה

נניח שאנו רוצים למלא את המסך ב-256 נקודות (שתופסות כ-6.5 שורות). על מנת לעשות זאת, עלינו לפתוח בבית מספר 40000 (קרי: $9C40) ולהציב בתוכו את הערך: 14 (קרי: $E). אחר כך, לגשת אל בית 40001 וגם בו להציב את הערך 14. כך גם בבית 40002, 40003 וכן הלאה, עד בית 40255 (או: $9D3F).

הקלדת 256 זוגות של פקודות: LDA ו-STA תהייה כמובן משימה מייגעת ביותר ועלינו למצוא דרך אלגנטית וקצרה לעמוד במשימה.

? עד כה למדנו לקדם את ערכו של בית יחיד בזיכרון באמצעות רגיסטר X או רגיסטר Y, אולם האם אפשר להיעזר במונה גם כדי להתקדם בית אחר בית בתוך הזיכרון?

התשובה היא כן.

כל שעלינו לעשות הוא לשנות את המבנה של הפקודה STA ובמקום ערך קבוע, לשים בה **ערך מתחלף**.

נניח שהשתמשנו ברגיסטר X כמונה. נשתמש גם באוגר ונפקוד כך:

```
STA  9C40,X
```

שים לב כי נעשה כאן שימוש בשני רגיסטרים יחדיו: רגיסטר A (האוגר) נושא את <u>הערך</u> שאותו נפקוד לתוך תא בזיכרון, ואילו רגיסטר X משמש כדי להתקדם מכתובת אחת בזיכרון אל הכתובת העוקבת אחריה.

■ כתוב כעת תוכנה קצרה שתמלא כ-6.5 שורות רצופות בסימן הנקודה (כלומר שתצייר על המסך 256 נקודות).

[תשובה 11 בעמוד 140]

■ שנה את התוכנה כך שתמלא את 6.5 השורות הראשונות של המסך בסדרה של כל 256 התווים (לפי ערכי ה-ASCII שלהם)

[תשובה 12 בעמוד 140]

המונה הישיר

עד כה עסקנו בזוג הפקודות: INX ו-DEX המקדמות (קדימה או אחורה) את רגיסטר X, ובזוג הפקודות: INY ו-DEY המקדמות (קדימה או אחורה) את רגיסטר Y.

יש מקרים בהם שלוש ה"ידיים" שלנו עסוקות בפעולות אחרות, ונדרשת ברקע פעולה פשוטה נוספת הכרוכה בהגדלה או בהקטנה של **בית** מסוים בזיכרון.

לשם כך, נוכל להיעזר בזוג הפקודות: INC ו-DEC

השימוש בפקודות אלו פשוט ביותר, ועל מנת להדגים את אופן פעולתן, נפתח תוכנית קצרה שמקדמת (קדימה או אחורה) את כל תווי המחשב. הקשה על החץ הימני, תתקדם אל התו הבא, ואילו הקשה על החץ השמאלי, תחזור אל התו הקודם.

לצורך התרגול, נניח שעומד לרשותנו רק רגיסטר פנוי אחד – רגיסטר X

■ נסה לכתוב בעצמך את התוכנה אך בכל מקרה עיין גם בהסבר שלי.

בשלב הראשון, נבנה את הלולאה המרכזית, אשר בוחנת את מצב לוח המקשים ויודעת לדלג החוצה, בכל אימת שמקישים על אחד מן המקשים הרצויים (חץ ימינה או חץ שמאלה)

```
10     *=$600
20 LOOP LDA $2FC
30     CMP #$7
40     BEQ FWD
50     CMP #$6
60     BEQ BWD
70     JMP LOOP
```

בשלב השני, נבנה את שתי אפשרויות הדילוג מן הלולאה: (1) התקדמות קדימה (אשר הגדרנו בתווית: FWD), ו-(2) התקדמות אחורה (אשר הגדרנו בתווית: BWD).

לצורך ההתקדמות (קדימה או אחורה) של תווי המחשב, ניעזר בפקודות: INC ו-DEC

```
100 FWD
110     INC $9C40
150     JMP LOOP
200 BWD
210     DEC $9C40
250     JMP LOOP
```

■ הרץ את התוכנה.

האם אתה מרוצה?

סביר להניח שלא. במקום לקדם את התווים צעד-אחר צעד, הקשה על כל אחד ממקשי החצים גורמת לתווים להתרוצץ בקצב מסחרר.

שים לב, בית $2FC המנטר את לוח המקשים, למעשה "זוכר" את המקש האחרון שהוקש. כדי לתקן את התוכנה, עלינו לגרום לבית הזה "לשכוח" את המקש שנלחץ, כלומר להציב בו בכל פעם את הערך 255 (או $FF), המייצג את המצב שבו אף מקש אינו לחוץ.

נוסיף אם כך את הפקודות הבאות:

```
120     LDA  #$0
130     STA  $2FC
220     LDA  #$FF
230     STA  $2FC
```

■ הרץ שוב את התוכנה ובדוק שהיא פועלת לשביעות רצונך.

תוכל להוסיף לה גם אופציית יציאה (למשל על ידי הקשה על מקש ה-ESC, שערכו: 28, כלומר: $1C)...

להטוטי ידיים

בשלב זה של לימוד יסודות האסמבלי, אתה כבר יודע כי יש הבדלים בין שלושת הרגיסטרים השונים. רגיסטר A (האוגר) למשל, ניכר ביכולותיו החישוביות. הרגיסטרים X ו-Y לעומת זאת, הם היחידים המאפשרים פקודות

מנייה. זאת ועוד, בפרקים הבאים נלמד על משימות נוספות הייחודיות לכל אחד משלושת הרגיסטרים.

? עכשיו נשאלת השאלה מה קורה אם טענו ערך אל תוך רגיסטר X למשל, וברצוננו להפעיל עליו פעולה חשבונית? האם ניתן להעביר מידע מרגיסטר אחד למשנהו?

התשובה היא כן.

בדיוק כפי שאתה יכול להעביר את העיפרון מיד ימין אל יד שמאל ולהיפך, כך גם באסמבלי קיימות ארבע פקודות ייחודיות להחלפה בין הרגיסטרים.

הפקודה **TAX** (קיצור של: Transfer A to X) מעתיקה את הערך שאוחז רגיסטר A לרגיסטר X.

הפקודה **TXA** (קיצור של: Transfer X to A) מעתיקה את הערך שאוחז רגיסטר X לרגיסטר A.

הפקודה **TAY** (קיצור של: Transfer A to Y) מעתיקה את הערך שאוחז רגיסטר A לרגיסטר Y.

הפקודה **TYA** (קיצור של: Transfer Y to A) מעתיקה את הערך שאוחז רגיסטר Y לרגיסטר A.

? האם ישנן פקודות להחלפה בין רגיסטר X ו-Y, ולהיפך?

התשובה היא לא. אך אם מתעורר צורך בכך, ניתן לעשות זאת בשני שלבים: ראשית, להעתיק את הערך שאוחז רגיסטר X למשל, אל רגיסטר A – באמצעות הפקודה **TXA**, ואחר כך להעתיק את הערך שאוחז רגיסטר A לרגיסטר Y באמצעות הפקודה **TAY**.

סיכום הפרק

בפרק זה למדנו לשלב יחדיו את שלושת הרגיסטרים: A, X ו-Y.

סקרנו את הפקודות: **LDX** ו-**LDY**, המקבילות לפקודה **LDA**, וכן את הפקודות: **STX** ו-**STY** המקבילות לפקודה **STA**.

גם לפקודת ההשוואה **CMP** (המשמשת לבחינת תנאי דילוג מלולאה) ישנה מקבילה עבור רגיסטר X – וזוהי הפקודה: **CPX**, וכן מקבילה עבור רגיסטר Y – וזוהי הפקודה: **CPY**.

בהמשך למדנו להעתיק מידע בין רגיסטר לרגיסטר באמצעות הפקודות: **TAX**, **TXA**, **TAY**, **TYA**.

הנושא המרכזי שסקרנו בפרק זה הוא מנייה, ולצורך כך למדנו להיעזר ברגיסטרים X ו-Y. הפקודות: **INX** ו-**INY** משמשות למנייה בסדר עולה, ואילו הפקודות **DEX** ו-**DEY** –משמשות למנייה בסדר יורד.

לבסוף, למדנו צורת שימוש חדשה לפקודה **STA**:

STA תא זיכרון**, X**

או:

STA תא זיכרון**, Y**

צורת שימוש זו, משלבת כוחות בין האוגר לבין אחד הרגיסטרים ומאפשרת לפקוד ערך יחיד (או סדרתי) לתוך **רצף** של כמה תאי זיכרון.

פרק ו
צעד לפנים וצעד לאחור

בפרק הקודם נעזרנו בשעון הפנימי של המחשב כדי לשלוט בקצב תחלופת צבעי רקע המסך. לשם כך יצרנו, כזכור, "לולאת השהייה".

לעיתים נרצה להפעיל לולאת השהייה שכזו בכמה מקומות בתוכנית שלנו. נוכל כמובן להקליד שוב ושוב את אותו רצף פקודות נדרש בכל אימת שנרצה, אך יעיל הרבה יותר יהיה להגדיר פעם אחת ויחידה את "לולאת ההשהיה", ולפנות אליה שוב ושוב כ**סאב-רוטינה**.

היזכר בזוג הפקודות **GOSUB** ו-**RETURN** בשפת הבייסיק. פקודות אלו מאפשרות לך לדלג, מכל מקום בתוכנה, אל נקודה ספציפית, ואחר כך לשוב אל **אותו המקום** אשר ממנו דילגת.

למעשה, הפנייה לסאב-רוטינה באסמבלי, דומה מאוד במהותה לשימוש בפקודות **GOSUB** ו-**RETURN**. על מנת לדלג אל **התווית** שממנה מתחילה הרוטינה נשתמש בפקודה **JSR** (קיצור המילים: Jump to SubRoutine) ועל מנת לחזור בחזרה אל המקום ממנו דילגנו, נשתמש בפקודה **RTS** (קיצור המילים: ReTurn from Subroutine)

על מנת להדגים את צורת העבודה עם רוטינה, ניעזר שוב בדוגמת ריצוד צבעי רקע המסך.

■ הקלד את השורות הבאות:

```
10      *=$600
20      LDX #$0
30 LOOP INX
```

```
40      STX  $2C8
60      JMP LOOP
```

■ הרץ את התוכנה.

כצפוי, צבעי רקע המסך מבזיקים בקצב מסחרר, וייפסקו רק כשתקיש על RESET.

נבנה עתה רוטינת "לולאת השהייה" ונגדיר לה תווית בשם WAIT:

```
100 WAIT LDY #$0
110      STY $14
120 TIMER LDY $14
130      CPY #$F
140      BEQ END
150      JMP TIMER
160 END
```

על מנת לדלג אל הרוטינה, נשתמש בפקודה JSR:

```
50      JSR WAIT
```

על מנת לחזור מן הרוטינה אל המקום שממנו דילגנו, נשתמש בפקודה RTS:

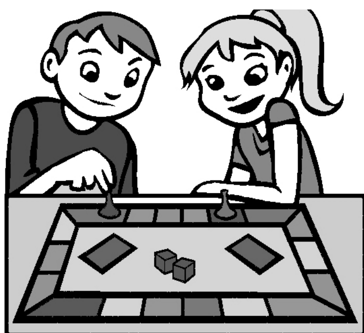
```
170      RTS
```

זכור כי תוכל לשוב ולדלג אל הרוטינה מתוך התוכנה בכל אימת שתחפוץ. כל שעליך לעשות לצורך כך הוא לפקוד: JSR WAIT.

דע לך כי קפיצות קדימה ואחורה (באמצעות הפקודה JSR למשל) מוגבלות בטווח המרחק שלהן. בין אם נשתמש בכתובות זיכרון כשלעצמן ובין אם נשתמש בתווית, המרחק המקסימלי שנוכל לקפוץ קדימה הוא 127 בתי זיכרון, ואילו המרחק המקסימלי שנוכל לקפוץ אחורה הוא 128 בתי זיכרון.

על מנת להבין את הלוגיקה הזו של הקפיצות קדימה ואחורה נדמיין לרגע משחק קופסה מוכר, מונופול למשל – אך עם חוקי משחק מעט שונים:

נניח שעל לוח משחק המונופול שלנו יש 40 משבצות (על חלקן רשומות כתובות כגון: ירושלים, תל אביב וכו'). עוד נניח כי על הלוח עומד שחקן אחד ויחיד (הפיון האדום למשל) והוא יכול להתקדם בכל תור צעד אחד קדימה, או לחלופין לקפוץ כמה צעדים – קדימה או אחורה – באמצעות הטלת שתי הקוביות (כשקפיצתו מוגבלת, מטבע הדברים, ל-12 צעדים לכל היותר).



בכל רגע נתון השחקן יודע כמובן היכן הוא עומד. אם למשל הוא עומד על משבצת: "רחוב העצמאות" בטבריה ומחליט להתקדם צעד אחד קדימה, הוא יגיע למשבצת: "רחוב הגליל".

לחלופין, הוא יכול להחליט להטיל את הקוביות ולקבל, לדוגמה 6, ואז לקפוץ שישה צעדים קדימה עד למשבצת: "רחוב דרך אילת" בבאר שבע, או שישה צעדים אחורה עד למשבצת: "חברת החשמל".

נחזור עתה אל עולם המחשב... המעבד – בדיוק כמוך, השחקן במשחק הקופסה מונופול – יודע בכל רגע נתון בדיוק היכן הוא נמצא, וזאת באמצעות "משתנה התמצאות" ייחודי הנקרא: Program Counter (או בקיצור: PC).

כל פקודה בשפת מכונה שהמעבד מבצע, מקדמת צעד אחד קדימה את אותו "משתנה ההתמצאות", PC. מצד שני, אם המעבד נתקל בפקודת קפיצה ייחודית (כגון: **JSR**) – הוא יודע, ממש כמו במקרה של הטלת הקוביות – לבצע דילוג קדימה או אחורה, וכמובן לעדכן במקביל כנדרש את אותו משתנה PC).

כיצד הוא עושה זאת? ממש כמוך, שחקן המונופול – המחשב <u>סופר</u> צעדים קדימה (חיבור) או סופר צעדים אחורה (חיסור).

מכיוון שרגיסטר PC, ממש כמו כל הרגיסטרים האחרים שהכרנו (A, X ו-Y) – מוגבל בגודלו ל-256 מקומות, גם הקפיצה מוגבלת לטווח זה.

? אבל רגע אחד... אמרתי כי הקפיצה קדימה מוגבלת ל-127 צעדים בלבד והקפיצה אחורה ל-128 צעדים בלבד. מדוע?

על מנת להבין זאת, עלינו לערוך היכרות עם דגל חדש – דגל המינוס (Negative flag) או בקיצור: דגל N.



דגל המינוס – הדגל N

התבונן ברצף הבינארי הבא: 11001100.

אתה כבר יודע כי 8 סיביות יכולות לייצג כל מספר עשרוני בין 0 ל-255, ואכן, אם נתרגם את הרצף הנ"ל למספר עשרוני, נקבל: 204.

ואולם, זהו רק **פירוש אחד** שניתן לתת לרצף בינארי מעין זה. פירוש מסוג אחר לחלוטין לרצף הבינארי 11001100 יכול להיות ייצוגה הגרפי של סדרת סיביות דולקות ומכובות (כפי שלמדת בפרק ד במחשבת 6):



וישנם כמובן גם פירושים נוספים...

אדריכלי המעבד 6502 התבוננו ברצפים הבינאריים בני 8 הסיביות ומצאו דרך יעילה לייצג באמצעותם לא רק מספרים חיוביים, אלא גם מספרים שליליים!

מכיוון שכל מה שמבחין בין מספר חיובי למספר שלילי הוא סימן אחד – של מינוס או של פלוס, אפשר להקצות סיבית אחת – השמאלית ביותר – לסימון "**כיוונו** של המספר על ציר המספרים" (0 על מנת לייצג מספר חיובי, ו-1 על מנת לייצג מספר שלילי).

בשבע הסיביות הנותרות ניתן לייצג כל מספר עשרוני בין 0 ל-127 (שהרי 7 סיביות דולקות נותנות את המספר העשרוני המקסימלי: 127).

את שיטת הייצוג הזו של מספרים שליליים נכנה מעתה: "שיטת שבע הסיביות".

נחזור שוב אל הרצף הבינארי: 11001100. ב"שיטת שבע הסיביות", הסיבית השמאלית ביותר מייצגת, כזכור, את סימן המינוס. את שבע הסיביות הנותרות נתרגם ונקבל את המספר העשרוני: 52(-)

חשוב להדגיש: הרצף הבינארי: 11001100 יכול לייצג הן את המספר העשרוני 204 (אם אנו עובדים בשיטת "שמונה הסיביות"), והן את המספר השלילי 52(-) (אם אנו עובדים בשיטת "שבע הסיביות").

שים לב, אם נחבר את המספר העשרוני 204, ואת ערכו המוחלט של המספר העשרוני השלילי 52(-) נקבל: 256.

ננסח זאת גם בהכללה: ערכו המוחלט של המספר העשרוני המיוצג בשיטת "שבע הסיביות", בצירוף ערכו המוחלט של המספר העשרוני המיוצג בשיטת "שמונה הסיביות" – מסתכמים תמיד ל-256!

להלן שתי דוגמאות לחישוב מהיר של המספר העשרוני השלילי המתקבל בשיטת "שבע הסיביות":

1. המספר העשרוני 196 מיוצג (בשיטת "שמונה הסיביות") על ידי הרצף הבינארי: 11000100
כדי למצוא את המספר העשרוני השלילי המקביל בשיטת "שבע הסיביות", נחשב: 256-196=60. נוסיף את סימן המינוס ונקבל: 60-

2. המספר העשרוני 240 מיוצג (בשיטת "שמונה הסיביות") על ידי הרצף הבינארי: 11110000
כדי למצוא את המספר העשרוני השלילי המקביל בשיטת "שבע הסיביות", נחשב: 256-240=16. נוסיף את סימן המינוס ונקבל: 16-

? כיצד יודע המחשב באיזו שיטת ייצוג רצינו להשתמש? האם התכוונו למספר 16- או למספר 240?

תשובה: הוא לא יודע! **המחשב מכיר אך ורק את הרצף הבינארי** 11110000. את המשמעות לרצף הזה נותנים המשתמשים – כלומר אנחנו.

בדיוק לשם כך משמש אותנו דגל השלילה N. אם האוגר נושא ערך הגדול מ-128 – הדגל N נדלק **אוטומטית** (כלומר ערכו נקבע על 1), ללמדנו כי הסיבית השמאלית ביותר נדלקה; ואם האוגר נושא ערך הקטן מ-128 הדגל N

נכבה **אוטומטית** (כלומר ערכו נקבע על 0), ללמדנו כי הסיבית השמאלית ביותר כבויה.

ולכן, ככל שנרצה לעבוד עם מספרים שליליים – כלומר בשיטת "שבע הסיביות", ניעזר בדגל N.

על מנת לבחון את מצבו של דגל N ישמשו אותנו שתי פקודות דילוג מלולאה:

BMI (Branch MInus) ו-**BPL** (Branch PLus)

עקרון הפעולה שלהן דומה מאוד לעקרון פעולתן של שתי הפקודות **BNE** ו-**BEQ**, אשר פגשנו לעיל, בפרק ג.

ועתה משימה קטנה...

דמיין שיש בידך שעון חול ובו 128 גרגירי חול, אשר גולשים מטה במשך 3 שניות. את חלקו העליון של השעון נכנה: "הצד החיובי" ואת חלקו התחתון נכנה: "הצד השלילי".



כאשר החול גולש מן "הצד החיובי" ומצטבר כולו ב"צד השלילי", שעון החול מתהפך על ראשו, ואז חלקו העליון הופך להיות "הצד השלילי", וחלקו התחתון הופך להיות "הצד החיובי", וחוזר חלילה.

כתוב תוכנית קצרה שמציגה את כיוונו של שעון החול בכל רגע נתון – אם חלקו העליון הוא "הצד החיובי", יופיע על המסך הסימן "+", ואם חלקו העליון הוא "הצד השלילי", יופיע הסימן "-".

כזכור, כתובת זיכרון 20 (או: $14), אחראית על "שניות" המחשב – ותוכל להיעזר בה כדי לבצע את המשימה.
לצורך כתיבת התוכנה, השתמש בשתי הפקודות: **BMI** ו-**BPL**.

[תשובה 13 בעמוד 141]

אגב, דוגמת שעון החול ממחישה כיצד מתנהגת סדרת המספרים החיוביים והשליליים, לפי שיטת "שבע הסיביות":

"הצד החיובי" של שעון החול הולך ויורד כך:

127, 126, ... 3, 2, 1, 0

או אז שעון החול מתהפך ו"הצד השלילי" הולך ועולה כך:

-128, -127, -126, ..., -3, -2, -1, 0

לאחר שסקרנו בהרחבה את דגל השלילה N, הגיע הזמן לעשות היכרות עם דגל חשוב נוסף – הרלוונטי לשיטת "שבע הסיביות" – והוא דגל הגלישה (Overflow flag), או בקיצור הדגל V.



דגל הגלישה - הדגל V

נניח שבחרנו לעבוד בשיטת "שבע הסיביות" ולערוך פעולות אריתמטיות בין מספרים חיוביים ושליליים.

בעמודים הקודמים ראינו כי בשיטת "שבע הסיביות" אנו מוגבלים למספרים בין 128- ל-127.

כמובן שמגבלה זו בעייתית ביותר וחייבים למצוא לה פתרון. יתרה על כן, לפעמים כפי שנראה גם חיבורם של מספרים "חוקיים" (העומדים במגבלה האמורה) עלול להסתכם בתוצאה שגויה!

נתבונן למשל בשלושה תרגילי חיבור עשרוניים:

(א) 7+(-12)=?

אם נתרגם את האיבר השני בתרגיל, לערכו (העשרוני) לפי שיטת "שבע הסיביות", נקבל: 7+[256-12]

התוצאה של תרגיל החיבור היא: 251, ואם נתרגם את המספר הזה לפי שיטת "שבע הסיביות", נקבל: [256-251]

שהם כמובן: 5-

ואכן: 7+(-12)=-5

(ב) (-8)+(-12)=?

נתרגם את שני האיברים לפי שיטת "שבע הסיביות" ונקבל:

[256-8]+ [256-12]

התוצאה של תרגיל החיבור הזה היא: 492.

כמובן שהאוגר לא יכול לשאת ערך החורג מ-256, ולכן דגל השארית (C) נדלק, וערכו של האוגר הופך: 236 (אחרי שהנחנו בצד 256 יחידות).

נחשב את התוצאה, לפי שיטת "שבע הסיביות", ונקבל: [256-236]

שהם כמובן: 20-

ואכן: (-8)+(-12)=-20

(ג) ?=110+50

שני האיברים 50, 110 הם מספרים חיוביים, ואולם התוצאה של התרגיל, לפי שיטת "שבע הסיביות" היא שלילית!! שהרי הערך 160 בשיטת "שבע הסיביות הוא למעשה 96(-).

האומנם 110+50=96 ?!

מבחינת המתכנת העובד בשיטת "שבע הסיביות" שגיאה שכזו עלולה להיות שערורייה של ממש!

אך אל דאגה... בדיוק לצורך כך בא לעולם דגל הגלישה, הדגל V. הדגל הזה נועד כדי להזהיר את המתכנת כי תוצאת התרגיל האריתמטי יצרה מספר גדול יותר מ-128 ולכן שינתה בו **שלא במכוון** את הסיבית השמינית.

? ומה המתכנת יכול לעשות במקרה שכזה?

לאכסן את התוצאה בשני בתים, במקום בבית אחד!

נחזור אל תרגיל החיבור האחרון ונדגים.

כדי לפתור את התקלה, שבה חיבור של שני מספרים חיוביים נותן מספר שלילי, יש **להפוך בחזרה** את הסיבית השמינית מ-1 ל-0. פעולה זו תגרע למעשה מן התוצאה את ערכה של הסיבית השמינית – הלוא הוא 128, כלומר התוצאה שתתקבל היא: 32=128-(110+50). עתה נדרש בית נוסף שיזכור את אותם 128 יחידות שגרענו מן התוצאה (קרי אותה סיבית שהפכנו).

ובסיכומו של דבר – בית אחד יישא את המספר 32, ובית נוסף יישא את המספר: 128, ויחדיו – את התוצאה: 160!

שים לב, כשמבצעים פעולת הוספה (ADC) בשיטת "שמונה סיביות" ומתקבלת תוצאה גדולה מ-255, נדלק כזכור הדגל C ונדרשים שני בתים כדי להחזיק את התוצאה. בדיוק באותו אופן, כשמבצעים פעולת הוספה (ADC) בשיטת "שבע סיביות" ומתקבלת תוצאה גדולה מ-127, נדלק הדגל V ונדרשים שני בתים כדי להחזיק את התוצאה.

דגל הגלישה V מזוהה עם שתי פקודות המאפשרות לדלג מלולאה בהינתן תנאי מסוים:

הפקודה BVS (קיצור המילים: Branch V Set) מאפשרת דילוג כאשר דגל הגלישה נדלק (כלומר כשערכו שווה ל-1)

הפקודה BVC (קיצור המילים: Branch V Clear) מאפשרת דילוג כאשר דגל הגלישה מכובה (כלומר כשערכו שווה ל-0)

בנוסף – בדומה לדגל השארית (C) ולדגל האפס (Z), גם את דגל הגלישה אפשר לכבות, וזאת באמצעות הפקודה CLV (קיצור המילים: CLear V).

לפני כל שימוש בשיטת "שבע סיביות" אמליץ לך לכבות את דגל הגלישה באמצעות פקודה הפקודה CLV.

■ ועתה משימה מאתגרת – בניית משחק זריזות ידיים!

כתוב תוכנה קצרה, שמאפשרת למשתמש להקיש במהירות 128 פעמים על מקשים שונים במקלדת, בטרם חולפות 15 שניות. בתום 15 שניות, אם השחקן נכשל במשימה, רקע המסך ייצבע באדום של מפסידים!

בבניית התוכנה, היעזר בפקודת הדילוג: BVS

נסה לכתוב את התוכנה בכוחות עצמך, אך בכל מקרה, עיין גם בהסבר שלי...

בשלב הראשון עלינו ליצור לולאה אשר תבחן בכל רגע נתון את מצבה של המקלדת, וכל אימת שהשחקן מקיש על מקש כלשהו, תתווסף יחידה אחת לערך השמור באוגר.

שים לב, הבית הבוחן את מצב המקלדת הוא כזכור: 764 (או: $2FC), וכפי שהסברתי בסוף פרק ה', הבית הזה חייב "לשכוח" בכל רגע נתון את ההקשה הקודמת, ולצפות להקשה חדשה.

הדגל שאנו בוחנים כל העת הוא דגל הגלישה (V) וראשית כל נזכור לאפס אותו.

על מנת שדגל הגלישה (V) יושפע משרשרת ההקשות, עלינו להבטיח כי הסיבית השמינית **תתהפך** בסופו של דבר מ-0 ל-1. לשם כך, **עוד בטרם תתבצע פעולת ההוספה הראשונה**, עלינו לוודא כי הסיבית השמינית מאופסת.

למעשה, כל שעלינו לעשות לצורך כך הוא להציב בראשית הדרך באוגר את הערך: 1 (או בשיטת הספירה הבינארית: 00000001). בתום סדרת ההוספות, כאשר הערך השמור באוגר יגיע ל-128 (או בשיטת הספירה הבינארית: 10000000) – הסיבית השמינית תתהפוך מ-0 ל-1, ואז דגל הגלישה יידלק.

להלן שורות הקוד לביצוע שלב א':

```
10    *=$600
20    LDX  #$FF
30    STX  $2FC
70    CLV
```

```
80      LDA #$01
90 LOOP LDX $2FC
100     CPX #$FF
110     BNE ADD
120     BVS END
160     JMP LOOP
170 ADD ADC #$1
180     LDX #$FF
190     STX $2FC
200     JMP LOOP
240 END
```

שים לב, שורה 110 בודקת את מצבו של דגל הגלישה (V). כאשר ערכו של האוגר גדול מ-128, הסיבית השמינית מתהפכת מ-0 ל-1, וגורמת להדלקת הדגל.

■ הרץ את התוכנית. הקש במהירות על מקשי המקלדת שוב ושוב עד שהתוכנה תיעצר. מדוד בעזרת שעון עצר בכמה זמן הצלחת להקיש 128 הקשות...

בשלב השני נוסיף למשחק את הגבלת הזמן. כזכור, אנחנו רוצים שהשחקן שלנו ינסה להשלים את 128 ההקשות שלו בפחות מ-15 שניות.

לצורך כך ניעזר, כרגיל, בשעון הפנימי של המחשב, אך הפעם בבית 19 (או: $13). ראשית, נאפס את שני הבתים, 19 ו-20 ואחר כך נמדוד 3 יחידות של התקדמות בבית מספר 19 – שהן שוות ערך לכ-15 שניות רגילות.

לבסוף, ניצור דילוג מן הלולאה אם המשימה לא הושלמה בזמן.

להלן שורות הקוד שיש להוסיף:

```
50     STX  $13
60     STX  $14
130      LDY  $13
140      CPY  #$3
150      BEQ  FAIL
210 FAIL LDX  #$37
230      STX  $2C8
```

■ הרץ את התוכנית.

האם הצלחת להקיש 128 הקשות לפני שרקע המסך נצבע באדום?

כאשר אנו מפתחים משחק, עלינו לדאוג לכך שהשחקן לא ינצל "פרצות" על מנת להשלים את המשימה בדרך הקלה. במקרה שלנו, השחקן יכול להניח את אצבעו על מקש מסוים ולא להרפות ממנו, ובאופן זה להשלים בלא מאמץ את 128 ה"הקשות".

■ נסה זאת בעצמך...

על מנת לתקן את ה"פרצה" הזו, עלינו להכניס שינוי בתוכנה. במקום להציב בכל רגע נתון את הערך 255 בתוך בית 764, עלינו לוודא כי כל הקשה תהייה אפקטיבית, אך ורק אם היא תהייה **שונה מן ההקשה שקמה לה**.

? כיצד מתכנתים זאת?

עלינו "להניח" בצד את המידע על ההקשה הקודמת וליטול אותו בהמשך לצורך השוואה עם ההקשה הבאה. דא עקה אין לנו ידיים פנויות כדי "להניח" משהו בצד. רגיסטר A עסוק במניית ההקשות; רגיסטר Y עסוק בפיקוח על הזמן החולף, ואילו רגיסטר X עסוק בניטור ההקשות על המקלדת...

נדרש אפוא מקום נוסף שבו אפשר לאחסן מידע וגם לשלוף אותו בקלות בעת הצורך.

אחת האפשרויות העומדות לרשותנו הוא להיעזר בבית אחד (או יותר) בזיכרון – בית שאינו בשימוש ולא יסכן אותנו בתופעות בלתי רצויות.

כזכור, אנו עובדים בדף מספר 6, כלומר ב-256 הבתים החל מ-1536 ועד 1791. נוכל להיעזר לצורך המשימה באחד הבתים האחרונים בטווח הזה – למשל 1790 (או: $6FE).

■ נסה לשנות בכוחות עצמך את התוכנה כך שבמקום לאפס בכל רגע את בית 764, תבחן אם המקש שהוקלד שונה מקודמו.
היעזר בבית 1790 כדי לאחסן בתוכו מידע, ואף לשלוף מתוכו את המידע ברגע הנכון.

[תשובה 14 בעמוד 141]

דע לך כי העיקרון לפיו "מניחים בצד" מידע ו"מושכים" אותו בהמשך, הוא עיקרון תכנותי חשוב באסמבלי ובשפת המכונה. ואולם, הדרך שבה השתמשנו בעיקרון הזה לא אלגנטית במיוחד, משתי סיבות לפחות:

ראשית, ככל שהתוכניות שתכתוב תהיינה ארוכות יותר ומורכבות יותר, יש חשש של ממש שהמידע ש"תזרוק" אי שם בתוך אחד מתאי הזיכרון יידרס

ויימחק. זאת ועוד – לעיתים יידרשו לך כמה וכמה תאי זיכרון שכאלה המשמשים לאגירת מידע ושליפתו.

שנית, קח בחשבון כי התכנות בשפת מכונה אינו אמור להיות תלוי בהגדרות המחשב הספציפי עליו אתה עובד. שפת המכונה היא כזכור השפה שבה "מדבר" המעבד 6502 – והמעבד הזה יכול להיות המעבד של מחשב האטארי שלך, אך גם המעבד של מחשב קומודור 64 למשל. ולכן אזור זיכרון פנוי לשימוש על חומרה אחת, יכול להיות אזור ששמור בו מידע חיוני בחומרה אחרת.

האם ישנו אזור מוגדר וייחודי, שהמעבד יודע לפנות אליו בשעת הצורך ולאחסן בתוכו מידע? אזור, אשר נוכל להיות בטוחים כי יוקצה למטרה הזו לא רק במחשב הספציפי שלנו, אלא בכל חומרה שיושב עליה מעבד 6502?

התשובה היא – כן.

המקטע המיוחד הזה של זיכרון לאחסון מידע נקרא "מחסנית" ובפרק הבא נעסוק בו בהרחבה.

פרק ז
מחסנית הכנס...

דמה בנפשך שאתה אוחז במחסנית ריקה של רובה (M16 לדוגמה), וממלא אותה אט-אט ב-30 כדורים. כל כדור שאתה מכניס אל תוך המחסנית דוחף את קודמו לאחור, עד אשר המחסנית מתמלאת לחלוטין ואין בה מקום לכדור נוסף.

עתה דמה בנפשך שאתה טוען את הרובה שלך במחסנית המלאה שהכנת; יורה את כל הכדורים בזה אחר זה; אחר כך שולף את המחסנית הריקה, ומתחיל למלאה מחדש בכדורים נוספים, וחוזר חלילה.



נקל להבין כי הכדור אשר תירה **ראשון**, הוא למעשה הכדור **האחרון** אשר הכנסת אל המחסנית, ואילו הכדור שתירה **אחרון**, הוא למעשה הכדור **הראשון** אשר הכנסת אל תוך המחסנית. עקרון זה של: "**ב**א **ר**אשון, **י**וצא **א**חרון" הוא עיקרון חשוב – ונזכור אותו בראשי התיבות: "ב.ר.י.א".

נסכם עד כאן את שלושת עקרונות המחסנית:

(1) למחסנית יש גודל מוגדר וסופי של מקומות פנויים.

(2) אפשר לרוקן את המחסנית (בחלקה או במלואה) ואחר כך לשוב ולמלאה בכדורים חדשים.

(3) המילוי והפריקה של מחסנית הוא לפי העיקרון ה"בריא" (בא ראשון, יוצא אחרון).

ושיהיה לכולנו לבריאות!

גם המעבד שלנו – 6502 – עובד עם מחסנית, שנקראת גם Stack. הוא יודע – בעזרת כמה פקודות ייחודיות – לאחסן בתוך המחסנית הזו מידע ולשלוף אותו בשעת הצורך.

במחשב האטארי שלך, המחסנית הזמינה לשימוש כוללת 256 בתים – המוקצים בטווח הזיכרון החל מכתובת 256 (או: $100) וכלה בכתובת 511 (או: $1FF). אזור זה של הזיכרון נהוג גם לכנות "דף 1" (כזכור, בדיוק לפי אותו העיקרון כינינו את טווח הבתים $600-$6FF בשם: "דף 6").

שתי הפקודות המרכזיות המשמשות אותנו לטיפול במחסנית הן: **PHA** ו-**PLA**

הפקודה **PHA** (קיצור המילים: PusH Accumulator) מכניסה את ערך שמחזיק האוגר, אל תוך המחסנית.

הפקודה **PLA** (קיצור המילים: PulL Accumulator) שולפת מתוך המחסנית את הערך (האחרון שהוכנס לתוכה) ומכניסה אותו אל תוך האוגר.

נניח לצורך הדוגמה שהכנסנו אל המחסנית – באמצעות הפקודה **PHA** – את הערך 1; אחריו את הערך 2; ולבסוף את הערך 3.

? מהו הנתון הראשון שייכנס לאוגר כשנפקוד: **PLA** ?

כמובן: 3, שכן לפי העיקרון ה"בריא", הנתון 3, אשר נכנס למחסנית אחרון, הוא זה שייצא ממנה ראשון.

■ נניח שאנו רוצים לשמור במחסנית שלנו את המילה "star", ובהמשך לשלוף אותה מן המחסנית ולהציגה על המסך.

ערכי האסקי של אותיות המילה הם: $73, $74, $61, $72

כתוב תוכנה קצרה שטוענת את המחסנית בארבעת ערכים אלו, ואחר כך שולפת אותם בזה אחר זה מתוך המחסנית ומציגה אותם זה לצד זה על המסך, כך שתיווצר המילה: "star".

זכור כי האוגר הוא היחיד מבין שלושת הרגיסטרים שאפשר לדחוף את ערכו לתוך המחסנית.

[תשובה 15 בעמוד 142]

? האם זכרת לעבוד לפי העיקרון ה"בריא"?

אם כן, זכית בכוכב (star), אך אם לא, קיבלת עכברושים (rats)...

עד כה עמדנו על הדמיון בין עקרון פעולת מחסנית הרובה, לבין עקרון פעולת מחסנית המחשב (Stack). ואולם, לצד הדמיון, קיים גם הבדל מהותי.

בעוד שהכדור המשתחרר מן המחסנית ונורה מלוע הרובה **נעלם** לבלי שוב, הרי ש"כדור" המידע הנשלף מן המחסנית אינו נמחק, אלא זמין לשימוש חוזר.

עתה נשאלת השאלה כיצד ניתן לגשת שוב אל אותו "כדור" מידע. הרי כל אימת שפקדנו על המחשב PLA, משכנו מן המחסנית "כדור" של מידע, ואגב כך הנענו את ה"קפיץ" של המחסנית קדימה, אל "כדור" המידע הבא.

? האם אפשר להחזיר את "קפיץ" המחסנית אחורה?

התשובה: בהחלט כן!

לשם כך בדיוק נועדו זוג הפקודות **TSX** ו-**TXS**

הפקודה **TSX** (קיצור של: Transfer Stack pointer to X) מורה למחשב לאחסן בתוך רגיסטר X את מיקומו המדויק של אותו "קפיץ" מחסנית, שמעתה ואילך נכנה אותו בשם "מחוון המחסנית" (Stack pointer).

הפקודה **TXS** (קיצור של: Transfer X to Stack pointer) מורה למחשב לשנות את מקומו של מחוון המחסנית, ולקבוע את המקום החדש לפי הערך השמור ברגיסטר X.

אדגים:

נניח שהמחסנית שלנו טעונה באותיות המילה: r, a, t, s. בטרם נתחיל למשוך את ארבע האותיות הללו מתוך המחסנית, נטען את מקומו של מחוון המחסנית אל תוך רגיסטר X באמצעות הפקודה **TSX**.

לאחר משיכת ארבע האותיות, נוכל לפקוד את הערך השמור ברגיסטר X, בחזרה אל מחוון המחסנית, באמצעות הפקודה **TXS** – וכך למעשה להשיב את המחסנית אל הנקודה ההתחלתית.

■ שנה כעת את התוכנה שכתבת, כך שתשלוף פעמיים מתוך המחסנית את המילה: "star", ותכתוב על המסך: star star

[תשובה 16 בעמוד 143]

עד כאן, לעת עתה, השימוש במחסניות. עוד נפגוש שתי פקודות נוספות המטפלות במחסנית בפרק ט העוסק בהפרעות (Interrupts)...

בינתיים נדגיש כי השימוש במחסניות יעיל מאוד בתכנות באסמבלי, אך הוא דורש כמה כללי זהירות:

(1) זכור כי גודלה של המחסנית מוגבל. יתרה על כן, חלק מן הבתים במחסנית מוקצים למשימות אחרות שקורות ברקע. לכן, שמור על טווח ביטחון ואל תאחסן כמויות גדולות מדיי של מידע במחסנית (בכל מקרה לא מעבר ל-200 בתים).

יחד עם זאת, תוכל כמובן להשתמש במחסנית לצרכים שונים באותה תוכנה כל אימת שתחפוץ, ולטעון אותה שוב ושוב במידע חדש.

דמה בנפשך כי מחסנית המחשב (בניגוד למחסנית הרובה) אינה סגורה בתחתיתה אלא פתוחה. ולכן כאשר תעבור את מכסת ה"כדורים" האפשרית (כזכור 256), המידע הישן יותר השמור במחסנית יתחיל "להיזרק" החוצה וייעלם, תוך שהוא מפנה מקום למידע חדש.

(2) זכור את העיקרון ה"ב.ר.י.א" – על מנת לאחזר מידע מתוך המחסנית, עליך להכניסו במהופך. מה שנכנס ראשון, יוצא אחרון. ולהפך.

(3) מכיוון שהמחסנית משמשת כאמור לא רק אותך, המתכנת, אלא גם את מעבד המחשב המבצע פעולות חיוניות ברקע, השתמש בפקודה: **TXS** בזהירות רבה.

בדוגמה האחרונה נעזרנו בפקודה **TXS** לצורך **משיכת** מידע מתוך המחסנית, וכיוונו מחדש את מחוון המחסנית לפי צרכינו. עשינו זאת באופן מבוקר, לאחר שווידאנו מראש היכן עמד מחוון המחסנית בראשית הדרך.

מצד שני, אם נשתמש בפקודה **TXS** באופן לא זהיר **ונדחוף** מידע אל תוך מקום במחסנית שכבר שמור בו מידע חיוני אחר, המחשב עלול "לקרוס" ותידרש לאתחל אותו מחדש.

פרק ח
מרחיבים את היריעה

אמור: שלום

בפרק ה לעיל סקרתי את הפקודות: `INX`, `INY`, `DEX` ו-`DEY`, אשר משמשות כמונה (עולה או יורד) עבור הרגיסטרים: X ו-Y.

הראיתי גם כיצד ניתן להיעזר בפקודות הללו על מנת להציב ערך מסוים לתוך רצף של תאי זיכרון.

ניזכר למשל בפקודה: `STA $9C40,X`

את רגיסטר X נוכל לקדם באמצעות הפקודה `INX` וכך למלא כ-6.5 שורות של המסך במופע חוזר של תו מסוים.

עד כאן טוב ויפה. אך מה אם נרצה לכתוב על המסך, לא רק רצף של תווים זהים (ואף לא סדרת תווים, עולה או יורדת לפי ערכי ASCII)?

מה אם נרצה לכתוב על המסך מילים, ואף משפטים שלמים כרצוננו – מבלי להתייגע בהקלדה של רצף הפקודות `LDA` ו-`STA` עבור כל תו ותו?

כדי לעשות זאת, נכניס לדיון פקודה חדשה של **כתבן האסמבלר**, ובנוסף – שימוש חדש בפקודה ישנה ומוכרת של שפת האסמבלי.

■ הקלד את התוכנה הבאה:

```
10     *=$600
20     LDY #$0
30 LOOP LDA DATA,Y
40     CMP #$0
50     BEQ END
60     STA $9C40,Y
70     INY
80     JMP LOOP
90 DATA
100    .BYTE $68, $65, $6C, $6C, $6F, $0
110 END
```

אסביר בפרוטרוט את שורות הקוד:

שורה 30 פותחת לולאה מותנית ומכניסה דבר מה אל תוך האוגר.

עד כה למדנו להכניס לתוך האוגר ערך יחיד מוגדר (למשל את הערך: #$40, או את הערך השמור בתא זיכרון: $14). ואולם, אין מניעה להכניס לתוך האוגר גם ערך כללי, השמור תחת תווית מסוימת.

נתבונן בשורות 90-100. שורה 90 פותחת בתווית, ומיד אחריה, בשורה 100, נפתחת רשימת ערכים בבסיס הקסה-דצימלי.

שים לב לפקודה BYTE הפותחת את שורת הקוד 100. בניגוד לפקודות בשפת האסמבלי עצמה, הפקודה BYTE היא פקודת עזר של **כתבן האסמבלר**. היא מאפשרת לפתוח רשימה מרובת ערכים – לכל צורך שהוא בתוכנה.

כשתעשה שימוש בפקודה **BYTE** זכור לפתוח את השורה בשלושה סימני רווח כמקובל, ואל תשכח להקליד לפני הפקודה את סימן הנקודה.

נחזור עתה לשורה 30. נקל לראות לפי מבנה הפקודה כי נעשה שימוש ברגיסטר Y לצורך מנייה.

? את מה בדיוק מונים?

את **האיברים** ברשימה השמורה תחת התווית.

האיבר הראשון ברשימה הוא $68. אחר כך, לאחר שערכו של רגיסטר Y מקודם ביחידה אחת (בשורת הקוד 70) ומתבצעת קפיצה אל שורת הקוד 30 (באמצעות הפקודה **JMP**, כמובן) – המחשב יודע לטעון אל תוך האוגר את הערך הבא ברשימה, הלוא הוא $68. ושוב, רגיסטר Y מתקדם, והאוגר נטען בערך הבא ברשימה – $6C. וחוזר חלילה.

? מה קורה כשהערך שהאוגר פוגש בו הוא $0?

ובכן לשם כך, נכתבו שורות הקוד 40 ו-50.

שורות אלו מורות למחשב לבצע דילוג מן הלולאה ולסיים את קריאת הרשימה.

סוף כל סוף, אחרי שמונה פרקים עמוסי-מידע, למדנו לכתוב על המסך "hello". ועדיין קיימת תחושת סירבול, וזאת מעצם העובדה שנדרש חישוב מייגע של ערכי אסקי של כל אות ואות, ועוד בשיטת הספירה ההקסה-דצימלית!

למזלנו, כתבן האסמבלר, פוטר אותנו ממשימה זו, ויכול לחשב עבורנו את ערכי האסקי כנגד כל אות ואות.

על מנת להיעזר בשירותי כתבן האסמבלר, כל מה שנדרש מאיתנו הוא להקליד – במקום את ערכי האסקי של האותיות – את המילה עצמה, כשהיא תחומה במירכאות.

■ החלף את שורה מספר 100 בשורה הבאה:

```
100    .BYTE "hello", $0
```

עכשיו הכול פשוט וקל!

אמור גם בעברית: שלום

במחשבת 6 "מסע אל זכרונו של המחשב" למדת לבנות בעצמך מערך סימנים חלופי הכולל את כל האותיות בעברית (אציע לך לשוב אל מחשבת 6 ולרענן את זיכרונך).

עתה יש בידיך את כל הכלים הדרושים לבניית מערך סימנים שכזה גם באסמבלי ובשפת מכונה!

כדי לגשת למשימה, אפרט את השלבים הנדרשים:

(1) בשלב הראשון, עלינו להעתיק את מערך הסימנים הקיים ב-ROM אל אזור ייעודי ב-RAM.

(2) בשלב השני עלינו להכניס את המידע הרלוונטי עבור כל האותיות העבריות, אל תוך מקום מתאים בזיכרון.

(3) בשלב השלישי, עלינו להודיע למחשב מהיכן לקרוא את מערך הסימנים החדש.

■ כתוב תוכנה קצרה אשר מעתיקה את מערך הסימנים הקיים ב-ROM אל אזור מקום חדש ב-RAM, ומכוונת את המחשב אל המיקום החדש.

שים לב, כל עוד אנו עובדים במצב הגרפי הפשוט של כתבן האסמבלר (גרפיקה 0), נוכל לבחור לצורך המשימה את האזור הסמוך ביותר לכתובת הראשונה המוקצה לזיכרון מסך זה (שהיא כזכור: 40000).

מכיוון שעבור מערך הסימנים כולו דרושים 1024 תאי זיכרון, אני ממליץ לך לפתוח בכתובת הזיכרון 38912 (או: $9800), כלומר לעבוד על ארבעת הדפים אשר החל מדף 152 ואילך.

אזכיר לך גם כי הבית המצביע על מיקום מערך הסימנים הוא: 756, והערך השמור בו הוא: 224.

[תשובה 17 בעמוד 143]

לאחר שיצרנו עותק חדש של מערך הסימנים במקום חדש בזיכרון, הגיעה העת לעצב את אותיות העברית.

את מלאכת העיצוב אשאיר לך, ואם כבר עמלת על המשימה במהלך לימוד מחשבת 6, תוכל כמובן להשתמש בנתונים שכבר חישבת. אל תשכח להמיר את הערכים לבסיס הספירה ההקסה-דצימלי!

יחד עם זאת, לצורך המשימה אציע להלן את העיצוב שלי לארבעת האותיות המרכיבות את המילה "שלום". הערכים להלן כתובים בבסיס הקסה-דצימלי.

האות ש:	$0, $92, $D2, $E6, $CC, $FC, $0, $0
האות ל:	$C0, $FC, $6, $6, $E, $3C, $0, $0
האות ו:	$0, $3C, $1C, $C, $C, $C, $0, $0
האות ם:	$0, $7E, $66, $66, $66, $7E, $0, $0

■ הוסף לתוכנה שכתבת את שורות הקוד הנדרשות לעיצוב ארבע האותיות הנ"ל.
לצורך התרגול, מקם את האותיות העבריות במקום האותיות abcd (ולא לפי סדר הופעתן על המקלדת).

[תשובה 18 בעמוד 144]

שמור את התוכנית שכתבת. עוד נחזור אליה בפרק י'.

עתה תוכל לעצב בעצמך את כל אותיות העברית ולמקם אותן לפי סדר הופעתן על גבי המקלדת.

לתכנות באסמבלי יש יתרון עצום על פני הבייסיק בכל הנוגע למהירות. הטעינה של כל מערך הסימנים החדש, כולל האותיות בעברית שתעצב – מתרחשת כהרף עין ממש!

פרק ט
סליחה שאני מפריע לך...

אין ספק כי הרכיב החשוב ביותר במחשב שלך הוא המעבד. אותה יחידה מרכזית אשר מתבצעים בה אלפי חישובים ומשימות בכל שנייה ושנייה. ואולם ברור כי המעבד אינו עומד בפני עצמו. על מנת שהמחשב יוכל לבצע את תפקידיו נאמנה, נדרשים גם אמצעי קלט (כגון המקלדת, כונן הדיסקטים) ואמצעי פלט (כגון: המסך, המדפסת). בנוסף, כבר התוודעת במחשבת 7 לשני רכיבים נוספים המצויים בתוך קרביו של המחשב – הלוא הם המיני מעבדים: ANTIC ו-GITA אשר אחראים על הגרפיקה ועל הצלילים.

על מנת לעבוד בסנכרון זה מול זה, נדרשת תקשורת של קבע בין המעבד המרכזי לבין כל אחד מהרכיבים האחרים. את התקשורת הזו נהוג לכנות גם בשם "הפרעה". זאת משום שאותם רכיבים למעשה "מפריעים" למעבד לבצע את משימותיו השוטפות, ומאלצים אותו להתפנות אליהם.

ככלל, ניתן לחלק את ההפרעות לשני סוגים: הפרעות שניתן להתעלם מהן לפי בקשה (Interrupt ReQuest או בקיצור IRQ) והפרעות שאי אפשר להתעלם מהן (Non Maskable Interrupts או בקיצור NMI). אסביר:

1. הפרעות מסוג IRQ

בפרק המבוא עשינו אנלוגיה בין מעבד המחשב לבין המוח האנושי. נשוב אל אותה אנלוגיה ונדמה עתה כי מוחנו – אותו מעבד רב-עוצמה – עסוק כעת

בביצוע משימה של זימרה. כדי להפיק את צלילי הזמר, מיתרי הקול בגרוננו מקבלים פקודות ("בשפת מכונה") להתכווץ ולהירפות, כך גם שרירי השפתיים והלשון וכיוצא באלו.

ואולם, לצערך (או שמא לשמחתך), אינך הדייר היחיד בבניין. בדירה שממולך מתגוררת שכנה שגם היא אוהבת לשיר, ואפילו במצו-סופרן! ובדיוק כעת היא החליטה לפצוח בזמרה ולשיר אריות אופראיות בקולי קולות.

זוהי אכן הפרעה של ממש! הפרעה שמקשה על המוח שלך לבצע את משימתו שלו ואולם, למזלך יש בכוחך להתעלם מאותה הפרעה בדרך פשוטה למדיי. תוכל לסתום את אוזניך באצבעותיך או בעזרת אטמי אוזניים!

שים לב. אמנם לא **עצרת** את ההפרעה מלהתרחש (כלומר לא השתקת את השכנה שלך בדרך מנומסת יותר או פחות), והיא ממשיכה להתקיים ברקע, אך אתה **בחרת להתעלם** ממנה ולצורך כך סתמת את אוזניך.

נשוב אל עולם המחשב, והנה גם כאן ישנן הפרעות שניתן להורות למעבד להתעלם מהן. אמנם לא תוכל – כמתכנת – למנוע מהן מלהתקיים, אך תוכל להורות למחשב להניח להן לפעול ברקע מבלי שיפריעו לו במשימותיו.

אחת הדוגמאות להפרעות כאלו – מסוג ה-IRQ – היא התקשורת שבין לוח המקשים לבין המעבד המרכזי. בעזרת הוראה פשוטה תוכל להורות למחשב להתעלם לרגע מן המקשים הנלחצים על המקלדת, ולשוב ולהתייחס אליהם ברגע אחר.



על מנת לעשות כן, אציג דגל נוסף המצטרף אל הדגלים שכבר פגשנו – והוא דגל ההפרעה (Interrupt flag) שמכונה בקיצור דגל I.

אם נרצה למנוע מהפרעה מסוג IRQ מלהפריע למעבד, נדליק את הדגל, ואם נרצה להשיב את המצב לקדמותו נכבה את הדגל.



זכור: הדלקת דגל I משולה לאטימת האוזניים בפני הפרעות.

? ואיך מדליקים ומכבים את הדגל I?

ובכן, הפקודה **SEI** (קיצור המילים SEt Interrupt) מדליקה את הדגל I (כלומר קובעת את ערכו כ-1).
הפקודה **CLI** (קיצור המילים CLear Interrupt) מכבה את הדגל (כלומר קובעת את ערכו כ-0).

■ ועתה משימה קטנה.
כתוב תוכנית קצרה המאפשרת למשתמש להקיש על מקשים שונים במקלדת במשך שלוש שניות, ואגב כך לשנות את צבעי רקע המסך. אחר כך, מנע מן המשתמש למשך שלוש השניות הבאות את היכולת לשנות את צבעי המסך, וחוזר חלילה. לצורך המשימה השתמש בפקודות: **SEI** ו-**CLI**

[תשובה 19 בעמוד 145]

2. הפרעות מסוג NMI

נחזור שוב אל האנלוגיה שבה פתחנו ונשוב אליך, הזמר הנלהב שהשכיל להתגבר על קולות הזימרה של שכנתו, וסתם את אוזניו.

עתה, כשאיש אינו מפריע לך עוד, המוח שלך יכול לפקוד על מיתרי הקול להתכווץ ולהימתח ולהפיק את צלילי הזמר הרצויים. ואולם האם משאביו של המוח אכן יכולים להיות מוקדשים כל כולם למשימת השירה?

בוודאי שלא! שהרי, עם כל הכבוד למשימה, הגוף צריך גם לנשום מדי פעם בפעם! אחת לכמה שניות נשלח אפוא מעין שדר בהול אל הסרעפת להתכווץ ולמלא את הריאות באוויר. אותו שדר דוחק הצידה כל משימה אחרת ומתבצע באופן מיידי (אם כי זמר מקצועי יודע להתאים בין צלילי הזמר שגרונו מפיק לבין נשימותיו).

נחזור עתה אל עולם המחשב. והנה, גם בתוך ה"מוח" של המחשב נמסרים אחת לזמן מה, ובאופן מחזורי, שדרים בהולים אל המעבד. שדרים אלו דוחקים הצידה כל משימה אחרת שמתבצעת ברקע, והם מתבצעים באופן מיידי ובלתי ניתן לביטול.

? מי מפריע למעבד ומדוע?

אחת ההפרעות המרכזיות מן הסוג הזה, שבלתי ניתן להתעלם מהן (NMI) הוא הפרעת ריענון ציור המסך.

התבונן על מסך הטלוויזיה שלך. התמונה שעל המסך נראית לך קבועה ורציפה, אך דע לך שכל העת – מרגע שהדלקת את המחשב ועד אשר תכבה אותו – טורח המעבד, אחת לזמן מה, לצייר מחדש את כל המסך, מראשו ועד סופו. הוא עושה זאת שוב ושוב בקצב מסחרר (של חמישים פעמים בשנייה), וזאת על מנת לתת לעין האנושית תחושה של רציפות.

הגורם ה"מפריע", קרי הרכיב אשר שולח אל המעבד את אותם שדרים בהולים ומחזוריים הוא כמובן ה-ANTIC, אותו מיני-מעבד גרפי האחראי על כל מה שמתרחש על מסך הטלביזה שלך. וכיצד מגיב המעבד כשמגיעה אליו אותה

"הפרעה"? ובכן, המעבד חדל מיד מכל משימותיו האחרות; מטפל ללא דיחוי במשימה שנמסרה אליו ומצייר מחדש את המסך; ורק אחר כך חוזר אל המשימות שהניח בצד.

? עתה נשאלת השאלה מדוע כל זה אמור בכלל לעניין אותנו? כלומר, אם אותה תקשורת בין המעבד המרכזי לבין ה-ANTIC מתרחשת באופן מחזורי ובלתי ניתן לעצירה או התעלמות, מדוע המתכנת צריך בכלל לעסוק בה?

ובכן, דע לך כי יש באפשרותך לנצל את אותה "הפרעה", **ולספח** לאותו "שדר בהול" כמה הוראות נוספות משלך. הוראות אלו יבוצעו באופן מחזורי על ידי המעבד, וממש כמו משימת "ציור המסך מחדש", יפעלו אף הן כל העת ברקע מבלי להשפיע על משימות אחרות שתורה למחשב לבצע.

את כל נושא "הפרעות ציור המסך" והשימושים המגוונים שאפשר לעשות באמצעותן נלמד ביחידת הלימוד הבאה: "מחשבת 9: תצוגת המסך". עתה נחזור לענייננו ונבחן הפרעה קטנה ייחודית נוספת.

הפקודה BRK

שני סוגי ההפרעות שהצגתי לעיל – IRQ ו-NMI, הן הפרעות שאחראים להן רכיבים **בחומרה** (כגון ה-ANTIC, לוח המקשים ועוד). בנוסף לאלו, ניתן לייצר הפרעה גם **מתוך התוכנה**, וזאת באמצעות הפקודה: **BRK**.

הפקודה **BRK** משמשת בעיקר כלי עזר תכנותי. היא מאפשרת למתכנת לבחון שלב מסוים בתוכנה ולתקנו במקרה הצורך.

■ הקלד את התוכנית הבאה הכותבת בראש המסך (באופן לא חכם במיוחד) את המילה "now".

```
10    *=$600
20    LDA  #$6E
30    STA  $9C40
40    ADC  #$1
50    STA  $9C41
60    ADC  #$8
70    STA  $9C42
```

■ עתה, הוסף פקודת עצירה שתגרום למחשב לכתוב את המילה: "no".

[תשובה 20 בעמוד 145]



דע לך כי הפקודה **BRK** מקושרת לדגל נוסף, הדגל B. כל עוד התוכנה פועלת ברקע, דגל B נותר מכובה, ורק כאשר התוכנה נעצרת (או מסיימת את פעולתה) הוא נדלק.

שים לב, דגל זה שונה באופיו מן הדגלים האחרים שהכרנו, במובן זה שלא מקושרות אליו פקודות הדלקה, פקודת כיבוי או פקודת התנייה.

מצעד הדגלים

עד כה פגשנו שישה מתוך שמונת הדגלים שהמעבד המרכזי 6502 נעזר בהם. כדי להשלים את התמונה אציג בקצרה את השניים האחרונים:

(1) הדגל העשרוני – דגל D.



כידוע, שפת המכונה מבוססת על שיטת הספירה הבינארית, ואילו שפת האסמבלי מבצעת את חישוביה לפי שיטת הספירה ההקסה-דצימלית. נוכחנו

כבר כי הייצוג של סיביות בינאריות כמספרים הקסה-דצימליים הוא נוח ויעיל במיוחד. ואולם, אם בכל זאת נרצה לייצג (באופן בזבזני ומסורבל) את הסיביות הבינאריות כמספרים עשרוניים, יש דרך לעשות כן.

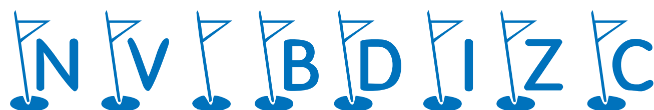
הפקודה **SED** (קיצור המילים: SEt Decimal) מדליקה את הדגל D ומורה למעבד לבצע את החישובים האריתמטיים בבסיס הספירה העשרוני.

הפקודה **CLD** (קיצור המילים: CLear Decimal) מכבה את הדגל ומשיבה את המצב לקדמותו, קרי: החישובים האריתמטיים מתבצעים בבסיס הספירה ההקסה-דצימלי.



(2) דגל ריק – ללא כל שימוש – שתמיד יימצא במצב דולק.

אלו הם אפוא שמונת הדגלים, לפי סדרם:



- הדגל N – דגל השלילה המקושר לפקודות **BMI** ו-**BPL**
- הדגל V – דגל הגלישה המגיב לפקודה: **CLV**, והמקושר לפקודות: **BVS** ו-**BVC**.
- דגל ריק שתמיד דולק!
- הדגל B – דגל העצירה המושפע מפקודת העצירה **BRK**.
- הדגל D – הדגל העשרוני המושפע מן הפקודות **SED** ו-**CLD**.
- הדגל I – דגל ההפרעה המושפע מן הפקודות **SEI** ו-**CLI**.

- הדגל Z – דגל האפס המגיב לפקודת ההשוואה CMP (כמו גם מקבילותיה: CPX ו-CPY), והמקושר לפקודות BEQ ו-BNE.
- הדגל C – דגל השארית המושפע מן הפקודות: CLC ו-SEC, מגיב לפקודות ההשוואה: CMP, CPX ו-CPY, כמו גם לפקודות החיבור והחיסור ADC ו-SBC, ולבסוף, מקושר לפקודות BCS ו-BCC.

דע לך כי לסידור הדגלים ישנה משמעות, במובן זה שאסופת 8 הדגלים מהווה למעשה בית אחד – יחידה אחת של שמונה סיביות.

על מנת להדגים את העניין, נבחן שני מצבים:

מקרה א: דגלי השלילה דולק, דגל הגלישה כבוי; דגל העצירה דולק; הדגל העשרוני כבוי; דגל ההפרעה כבוי; דגל האפס ודגל השארית דולקים.

במקרה זה נוכל לסמן את מצב הדגלים כך:

N V B D I Z C

(שים לב: הדגל השלישי משמאל הוא דגל ריק שכאמור דולק תמיד)
נוכל לחשב (לפי "שיטת המשקולות") ולראות כי ייצוג זה של 8 הדגלים נותן את המספר העשרוני 179, או המספר ההקסה-דצימלי $B3.

מקרה ב: דגלי השלילה כבוי, דגל הגלישה כבוי; דגל העצירה דולק; הדגל העשרוני כבוי; דגל ההפרעה כבוי; דגל האפס ודגל השארית כבויים.

במקרה זה נוכל לסמן את מצב הדגלים כך:

N	V		B	D	I	Z	C
		■	■				

נוכל לחשב (לפי "שיטת המשקולות") ולראות כי ייצוג זה של 8 הדגלים נותן את המספר העשרוני 48, או המספר ההקסה-דצימלי $30.

התבונן עתה על שורת הסיכום המוצגת אחרי הרצת התוכנה האחרונה:

```
0612    A=77    X=00    Y=00    P=30    S=00
```

האם הפריט החמישי, P=30 נראה לך מוכר?

ובכן המספר ההקסה-דצימלי $30 (או 00110000 בשיטת הספירה הבינארית) הוא בדיוק תמונת מצב הדגלים המאפיינת את הרגע בו התוכנה האחרונה שהרצת הסתיימה.

וזו העת לעשות היכרות עם רגיסטר נוסף (לצד שלושת ה"ידיים" A, X, Y), והוא הרגיסטר P. רגיסטר מיוחד זה – קיצור המילים: Processor status – מחזיק את ערכם של כל 8 דגלי המעבד, בכל רגע נתון.

בסעיף הבא נראה כיצד נוכל לעשות שימוש ברגיסטר זה.

לא משאירים עקבות

נניח כי עליך לכתוב מקטע קוד קצר הרושם את המילה no בפינה השמאלית של המסך, אגב שימוש חד פעמי בפקודה **LDA**.

שורות הקוד שתכתוב עשויות להיראות כך:

```
10    *=$600
60    LDA  #$6E
70    STA  $9C40
80    ADC  #$1
90    STA  $9C41
```

עתה, נניח שהמחשב היה עסוק במשימה אחרת לפני שפנה לבצע את מקטע הקוד שכתבת – משימה של חישוב כגון זה:

```
30    LDA  #$F0
40    ADC  $80
```

פעולת חיבור שכזו (בשורות 30-40) גורמת לשינוי מצבו של דגל השארית C, ואגב כך משבשת את הביצוע התקין של המקטע שכתבת (אם אינך זוכר מדוע, חזור ועיין בפרק ג׳ לעיל). וכך, במקום המילה no, יתקבל רצף האותיות: np.

■ נסה!

כדי לתקן את ה״תקלה״, תוכל כמובן להשיב את המצב לקדמותו ולכבות את דגל השארית C באמצעות הפקודה **CLC** (בשורה 50 למשל). כך אמנם עשית בפרק ג׳ לעיל. ואולם, ככל ששורות הקוד שתכתוב יהיו מורכבות יותר

תזדקק לשיטה יעילה יותר להשיב את המצב לקדמותו בשעה שהמחשב עובר ממקטע קוד אחד למשנהו. יתרה על כן, ביחידת הלימוד הבאה, כשנעסוק בהפרעות "ריענון המסך" ניתקל במצבים בהם כלל לא נוכל לדעת איזה שינוי מתחולל במצבם של הדגלים אגב המעבר ממקטע קוד אחד למשנהו, ולפיכך גם לא נדע אילו דגלים יש להדליק מחדש, ואילו לכבות, כדי להשיב את המצב לקדמותו.

למזלנו ישנה דרך יעילה להתמודד עם מצבים ממין זה, וכדי להסבירה, נדמיין סיטואציה מתחום שונה לחלוטין: דמה בנפשך שמאפשרים לך להיכנס לחדר, לעשות בתוכו כאוות נפשך ובתנאי שבצאתך ממנו תשיב את המצב לקדמותו, כלומר לא תשאיר אחריך שום עקבות! מה שעשוי להועיל לך ביותר הוא מצלמה, שכן אם תצלם את מצבו ההתחלתי של החדר לפני שנכנסת אליו, תוכל, ממש לפני צאתך משם, להתבונן בתמונה ולתקן על פיה את כל מה ששינית.

האם קיימת דרך "לצלם" את מצבם של הדגלים לפני ביצוע משימה מסוימת, ולקבוע מחדש את מצבם, עם סיומה?

התשובה היא כן. באמצעות שתי הפקודות **PHP** ו-**PLP**

הפקודה **PHP** (קיצור המילים: PusH Processor status) דוחפת את הריגסטר P (שהוא כאמור תמונת מצבם של כל 8 הדגלים) אל תוך המחסנית.

הפקודה **PLP** (קיצור המילים: PulL Processor status) עושה את הפעולה הנגדית – שולפת מן המחסנית את תמונת המצב השמורה של הדגלים וטוענת אותה אל תוך רגיסטר P. באופן זה היא קובעת מחדש את מצבם של כל הדגלים.

■ הוסף לתוכנית הקצרה את הפקודות הנדרשות על מנת שהחישוב המתבצע בשורות 20-30 לא ישפיע על מצבם של הדגלים בביצוע המשימה הבאה שבשורות 60-90.

[תשובה 21 בעמוד 145]

והפקודה שלא נלמד...

עד כה עברנו על פקודות רבות באסמבלר ובפרקים הבאים נסיים את המלאכה.

באסמבלר (גרסת המעבד 6502), ישנן 56 פקודות, ועל אחת מהן אני נאלץ לוותר ולהשאירה כ"טיזר" לספר הבא...

מדובר בפקודה `RTI` (קיצור המילים ReTurn from Interrupt), והיא קשורה במישרין להפרעות מסוג NMI.

בסעיף הקודם ראינו כי לעתים נידרש לצאת מהפרעה "מבלי להשאיר עקבות", והפקודה `RTI` יודעת לעשות זאת בכל הנוגע להפרעות מסוג זה. אך כל זאת – ביחידת הלימוד הבאה!

פרק י
האוגר מתנועע ומכפיל את משקלו!

בפרק ג הכרנו שתי פקודות חשבוניות שניתן להפעיל על האוגר: חיבור (ADC) וחיסור (SBC). בשלב מתקדם זה של הלימוד, ראוי שנתייחס גם לשתי הפעולות החשבוניות הנוספות – הלוא הן כפל וחילוק.

ראשית, אדגיש כי לא קיימות באסמבלי פקודות המאפשרות לבצע במישרין תרגילי כפל וחילוק כמו: 8*10 , 16/2. ואולם חישובים כגון אלו נוכל לערוך באמצעות שימוש מושכל בזוג הפקודות: ASL ו-LSR.

1. הפקודה ASL

התבונן במספר הבינארי: 00000001 (השווה למספר העשרוני: 1)

? מה יקרה אם נזיז את הספרה 1 מקום אחד שמאלה, כך: 0000010 ?

המספר כמובן יגדל ונקבל את המספר העשרוני: 2.

נמשיך ונזיז את הספרה 1 שלב אחר שלב שמאלה, ונקבל את התוצאות הבאות (משמאל למספר הבינארי מצוין הערך העשרוני):

1	00000001
2	00000010
4	00000100
8	00001000
16	00010000
32	00100000
64	01000000
128	10000000

נקל לראות כי כל הזזה שמאלה של הספרה 1, הגדילה את ערכה פי שניים.

תוצאה דומה נקבל כמובן גם עבור הזזה של כמה ספרות.

ניקח למשל את המספר הבינארי 00010110 (השווה למספר העשרוני 22).

אם נזיז את כל הספרות מיקום אחד שמאלה, נקבל: 00101100 (השווה למספר העשרוני 44).

נוכל לסכם ולומר: על מנת להכפיל ערך מסוים פי שניים, נוכל להמירו לשיטה הבינארית ולהסיט את הספרות שלו מקום אחד שמאלה.

בדיוק לצורך זה משמשת הפקודה ASL (קיצור המילים: Arithmetic Shift Left). היא מסיטה מקום אחד שמאלה את כל שמונה הסיביות של המספר הבינארי.

■ הקלד את שורות הקוד הבאות:

```
10      *=$600
20      LDA  #$10
30      ASL  A
```

הרץ, ובדוק את שורת המצב.

כצפוי, ערכו של האוגר הכפיל את עצמו והפך מ-$10 (או: 16 בשיטה העשרונית) ל-$20 (או: 32 בשיטה העשרונית).

שים לב למבנה הפקודה **ASL** – הפקודה מצפה למשתנה, שהוא במקרה זה האוגר.



האם ניתן לבצע את הפקודה **ASL** גם על הרגיסטרים האחרים – X או Y?

■ נסה!

המחשב מגיב בהודעת השגיאה: ERROR 5, שכן כפי שכבר למדנו בפרק ג, מבין שלושת הרגיסטרים, האוגר הוא היחידי ש"יודע" חשבון!

יחד עם זאת, את הפקודה **ASL**, כפי שנראה, אפשר להפעיל לא רק על האוגר אלא גם על משתנים אחרים.

אך ראשית, עליי לעמוד בהבטחתי ולהראות כיצד ניתן לפתור את תרגיל הכפל: 8*10

הפקודה **ASL** מסוגלת כפי שראינו להכפיל מספר פי 2. אם נפעיל אותה פעמיים רצופות, נוכל להכפיל את המספר (המקורי) פי 4, ואם נחוזר עליה שלוש פעמים ברצף, נוכל להכפיל את המספר (המקורי) פי 8.

על מנת להכפיל מספר ב-10, נוכל לחלק את המשימה לשניים:

(א) להפעיל על המספר המקורי שלוש פעמים את הפקודה **ASL**

(ב) להפעיל על המספר המקורי את הפקודה **ASL** פעם אחת.

ולבסוף, לחבר יחדיו את שתי התוצאות.

שהרי: $8*10=(8*8)+(8*2)$

? אבל רגע אחד. כיצד נוכל להניח בצד את תוצאת הביניים (8*8)?

ובכן, בדיוק לשם כך יכול לשמש אותנו **משתנה עזר** שנוסיף לתוכנה.

שים לב, משתנה עזר, בדומה לתווית, איננו חלק משפת המכונה אלא כלי עזר תכנותי שמספק לנו **כתבן האסמבלר**.

על מנת להיעזר במשתנה-עזר, ראשית כל עלינו להגדירו, והדרך הקלה ביותר לעשות זאת היא כך:

```
40  TEMP=0
```

שים לב, בדומה להגדרת תוויות, גם הגדרתם של משתני עזר תופרד ממספר השורה ברווח אחד. **רק לאחר שהגדרנו את המשתנה החדש**, נוכל לטעון לתוכו את ערכו של האוגר בעזרת הפקודה STA, כך:

```
50      STA  TEMP
```

■ כעת יש ברשותך את כל הידע לכתוב תוכנה קצרה באסמבלי שתחשב את התרגיל 10*8. בהצלחה!

[תשובה 22 בעמוד 145]

את הפקודה ASL תוכל לבצע גם על משתנה העזר שיצרת.

שנה למשל את שורות הקוד 60 ו-70, כך:

```
60      ASL  TEMP
70      ASL  TEMP
```

■ בדוק, וראה שאין כל הבדל בתוצאה!

עד כאן ראינו כיצד ניתן לפתור תרגילי כפל על האוגר, בהתבסס על הפקודה **ASL** אך כדי להשלים את התמונה, עלינו לבחון מצב נוסף – מצב שבו תוצאת המכפלה גדולה מ-255.

? נניח שהאוגר נושא את הערך $8C (או 140 בשיטה העשרונית) ומבצעים עליו את הפקודה **ASL**. איזו תוצאה תתקבל?

על מנת להשיב על השאלה נכתוב את המספר בשיטה הבינארית: 10001100.

עתה נסיט את כל הספרות מקום אחד שמאלה ונקבל:

00011000.

שים לב, הספרה השמאלית ביותר "עפה החוצה" והתוצאה המתקבלת היא $18 (או: 24)

הייתכן כי: 140*2=24 ?

בהחלט כן! משום שלמעשה הסיבית השמינית והאחרונה לא סתם "עפה החוצה", אלא נשמרת בתוך דגל השארית – C !

כאשר דגל השארית דולק, המתכנת (כלומר אתה) יודע כי תוצאת המכפלה חייבה "הנחה בצד" של 256 יחידות.

ואכן, אם נתחשב בערכו של הדגל, נקבל את פתרון התרגיל הנכון:

140*2 = 24+256 = 280

2. הפקודה LSR

אחרי שדנו בפעולת הכפל, נעבור לעסוק בפעולת החילוק, וזה המקום להציג את הפקודה **LSR** שמצבעת בדיוק את הפעולה ההפוכה מקודמתה – כלומר מסיטה את שמונה הסיביות של המספר הבינארי מקום אחד **ימינה**.

נתבונן למשל על המספר: 11001000 (השווה ל-200 בשיטה העשרונית).

אם נסיט את כל שמונה הספרות מקום אחד ימינה, נקבל את המספר: 01100100 (השווה ל-100 בשיטה העשרונית).

נוכל לסכם ולומר: על מנת לחלק ערך מסוים בשתיים, נמיר אותו לשיטה הבינארית ונסיט את הספרות שלו מקום אחד ימינה.

וזוהי תמצית פעולתה של הפקודה **LSR** (קיצור המילים: Logical Shift Right).

שים לב, פעולת החילוק באסמבלי, בניגוד לפעולת הכפל, אינה מאפשרת עבודה בכמה שלבים, וזאת על שום עצם טבעה של פעולת החילוק.

התבונן למשל על תרגיל החילוק: 15/3.

כמובן שזו תהייה שגיאה גסה לנסות לפרק אותו כך:

$$15/3 = 15/(2+1) \neq (15/2)+(15/1)$$

ולכן, בפקודה **LSR** תוכל להשתמש אך ורק כדי לחלק מספר ב-2, ב-4, ב-8 וכו'.

כמובן שאפשר לבנות רוטינה באסמבלי שתחשב תרגילי חילוק מכל סוג שהוא, אך במסגרת יחידת לימוד זו לא נעסוק בכך.

? עכשיו נשאלת השאלה, להיכן נעלמה הסיבית הראשונה (הימנית) אגב ביצוע הפקודה **LSR**?

ובכן – ודאי כבר ניחשת – הסיבית הימנית ביותר שהוסטה ימינה נשמרת בתוך דגל השארית – C !

? למה הדבר עשוי להועיל?

ובכן, נבחין בין שתי אפשרויות: אם הספרה (או הסיבית) הימנית ביותר במספר המקורי הייתה 0, משמע שהמספר היה זוגי, כלומר מתחלק בשתיים ללא שארית. מצד שני, אם הספרה (סיבית) הימנית ביותר במספר המקורי הייתה 1, משמע שהמספר היה אי-זוגי, כלומר לא מתחלק בשתיים.

כך למעשה, בכל הנוגע לפקודה **LSR** משמש אותנו דגל השארית C כאינדיקציה לזוגיות או אי-זוגיות המספר. דגל כבוי משמעו שהמספר המקורי היה זוגי. דגל דולק משמעו שהמספר המקורי היה אי-זוגי.

ועתה, משימה קטנה.

■ בפרק ד כתבת תוכנה קצרה שמחליפה את צבעי רקע המסך במהירות לפי קצב התחלפות "שניות" המחשב.

עכשיו אבקשך להוסיף כמה שורות קוד לתוכנה וליצור הפוגות בין ריצודי הצבע. כלומר, אחרי ריצוד של כ-5 שניות, תהייה הפוגה של חמש שניות וחוזר חלילה.

לצורך כתיבת התוכנה, היעזר גם בבית האוחז ב"דקות" המחשב. במשך פרק הזמן של כל "דקה" **זוגית**, המסך יחליף צבעים במהירות, ובמשך פרק הזמן של כל "דקה **אי זוגית**", תהייה הפוגה.

[תשובה 23 בעמוד 146]

3. ניפגש בסיבוב

הפקודות ASL ו-LSR שהכרנו בסעיפים הקודמים הן פקודות מתמטיות-לוגיות לכל דבר: ההזזה שמאלה או ימינה של שמונה הסיביות, אגב ההסתייעות בדגל השארית, מגדילה או מקטינה את המספר פי 2.

? האם אותה תזוזה של סיביות – ימינה או שמאלה – יכולה לשמש אותנו גם לצרכים אחרים שאינם מתמטיים?

בהחלט! שכן כפי שכבר הסברנו בפרק ו, שמונה הסיביות עשויות לייצג מספר בינארי, אך עשויות לייצג גם דברים אחרים.

נתבונן למשל על רצף הסיביות הבא: 00000011

כזכור, ניתן לייצג רצף זה גם כחלק (שמינית) מתוך תו מסוים.

עתה, נסה לדמיין מה יקרה אם נזיז שמאלה את שמונה הסיביות הללו, שוב ושוב, כמובן באמצעות הפקודה ASL.

אם חשבת על אנימציה, צדקת!

דא עקה, אופן פעולתה של הפקודה ASL יסיים את האנימציה הזאת מהר מאוד, שכן אגב הזזות שמונה הסיביות שמאלה, הסיבית השמאלית "נעלמת" (כזכור, אל תוך דגל השארית) ואילו הסיבית הימנית מתאפסת. וכך, אחרי שמונה חזרות על הפקודה ASL כל הסיביות של הבית יתאפסו!

? האם ישנה דרך לגלול את הסיביות שמאלה בתנועה מעגלית, כך שהסיבית השמאלית ביותר לא תיעלם אלא תופיע לה מן הצד השני, הימני (כדרך פעולתו של מנגנון מנעול הקומבינציה)?



התשובה היא: כן.

ולצורך כך משמשות אותנו שתי הפקודות **ROL** ו-**ROR**.

הפקודה **ROL** (קיצור המילים: ROtate Left) מזיזה את כל 8 הסיביות שמאלה, **בתנועה מעגלית**.

הפקודה **ROR** (קיצור המילים: ROtate Right) מזיזה את כל 8 הסיביות ימינה, **בתנועה מעגלית**.

■ ועתה משימה.

טען מחדש את התוכנה אשר בנית בפרק ח, המעתיקה את מערך הסימנים לאזור אחר בזיכרון. אחר כך שנה את התו "a" והדלק בשורה הראשונה המרכיבה אותו את שתי הסיביות הימניות ביותר, כך שיתקבל סימן המזכיר במעט את האות עם האקצנט: ā.

לבסוף, הוסף כמה שורות קוד אשר יזיזו שמאלה את שורת הסיביות הזו כך שאותו "אקצנט" ינוע שמאלה. חזור על הפעולה שוב ושוב על מנת ליצור תנועת אנימציה.

אל תשכח להוסיף לולאת השהייה קצרה בין כל הזזה שמאלה, על מנת לקבל אנימציה בקצב סביר.

[תשובה 24 בעמוד 146]

? האם אתה מרוצה מן התוצאה?

סביר להניח שלא.

במקום תנועה מעגלית של אותו "אקצנט" קיבלנו קו עילי שהולך ומתארך עד שהוא ממלא את כל השורה.

? האם יש לך מושג מה קרה כאן?

לפני שתמשיך לקרוא הלאה, חשוב...

ובכן, הבעיה נעוצה בעצם השימוש בפקודות **CPY** ו-**BEQ** (בלולאת ההשהיה). הפקודות הללו מעצם טבען עושות מניפולציה על דגל השארית C, אלא שדגל זה הוא חלק מהותי מפעולת הפקודה **ROR**.

על מנת לאפשר לפקודה **ROR** לבצע כיאות את הזזת הסיביות, חייבים להבטיח כי דגל השארית C לא יושפע מאף פקודה אחרת.

? כיצד עושים זאת?

בפרק הקודם למדנו להיכנס ולצאת מהפרעה מבלי להותיר עקבות. הגיע הזמן ליישם את מה שלמדנו.

■ הוסף כעת את שורות הקוד הנדרשות כדי לתקן את הבעיה.

[תשובה 25 בעמוד 147]

עתה אציע לך לשדרג את האנימציה וליצור לוכסן (/) שינוע שמאלה בתנועה אינסופית.

על מנת להימנע מן הצורך ליצור לולאה נוספת (שכזכור, משפיעה על דגל השארית C), אציע לך פשוט להקליד בזו אחר זו את הפקודות הנדרשות עבור כל אחת משמונה השורות הבונות את תו הלוכסן.

לפני שתיגש לכתוב את שורות הקוד, שרטט על נייר את כל התזוזות הנדרשות כדי לייצר תנועה נכונה.

[תשובה 26 בעמוד 147]

פרק י"א
לוגיקה מדליקה
(וגם מכבה)

פקודות לוגיות הן כלים רבי-עוצמה שעשויים לשמש אותנו במגוון משימות תכנותיות. בדומה לפקודות החשבוניות (דוגמת פקודת החיבור: **ADC**), גם הפקודות הלוגיות מופעלות על האוגר בלבד, והן משנות את הערך השמור בו.

תחום עניינה של הלוגיקה הוא האמת (TRUE) והשקר (FALSE) ובעולם ערכים בינארי שכזה, אך טבעי הדבר שבבואנו לעסוק בפקודות הלוגיות, נעדיף תמיד לעבוד **בשיטת הספירה הבינארית**.

כך, סיבית דולקת (1) תיוצג כבעלת ערך TRUE, ואילו סיבית כבויה (0) תיוצג כבעלת ערך FALSE.

תחום העיסוק בפקודות הלוגיות הוא למעשה תחום העיסוק במערכת יחסי הגומלין בין שתי סיביות (דולקות ו/או כבויות), כאשר את כל אחד משני "בני הזוג" נהוג לכנות בשם: "אופרנד".

בשפת האסמבלר של מעבד 6502 ישנן שלוש פקודות לוגיות – **ORA**, **AND** ו-**EOR**. להלן אציג כל אחת מהן.

1. הפקודה ORA

הפקודה הלוגית **ORA** (קיצור המילים: OR Accumulator) תיתן ערך TRUE (1), אם לפחות אחד האופרנדים הוא בעל ערך TRUE (1).

לפיכך, ״אפס או אחד״ יתנו אחד, וכמובן גם ״אחד או אחד״ יתנו אחד, ורק ״אפס או אפס״ ישאירו את הערך אפס.

את הנאמר במילים אפשר לסכם גם ב״טבלת אמת״ של הפקודה **ORA**:

	0	**1**
0	0	1
1	1	1

מהו אפוא, השימוש הפרקטי בפקודה **ORA** ?

היא מאפשרת **להדליק** סיביות כבויות.

אם למשל נרצה להדליק את הסיבית השמאלית ביותר בערך השמור באוגר, כל שעלינו לעשות הוא להיעזר בפקודה **ORA** ולפקוד לתוך האוגר את המספר הבינארי הבא: 10000000 (או #$80 בשיטה ההקסה-דצימלית):

```
ORA  #$80
```

כך, אם הסיבית השמאלית ביותר השמורה באוגר דולקת, היא תישאר דולקת, אך אם הסיבית השמאלית ביותר באוגר כבויה, היא תידלק.

ועתה משימה.

ידוע כי במחשב האטארי שלך ישנם 256 תווים – 128 תווים רגילים (כגון: **A**, **4**, **&**) ו-128 תווים נגטיביים (כגון: **A**, **8**, **=**). ערכי ה-ASCII של התווים הנגטיביים עוקבים כידוע אחר ערכי ה-ASCII של התווים הרגילים, בפער של 128 יחידות.

כתוב תוכנה קצרה באסמבלי הרושמת את המילה hello, ובמרווחים של כשתי שניות, הופכת את המילה לנגטיבית (לבן על גבי שחור) וחוזר חלילה.

לצורך התרגול השתמש בפקודה הלוגית **ORA**

[תשובה 27 בעמוד 148]

2. הפקודה **AND**

הפקודה הלוגית **AND** תיתן ערך FALSE (0), אם לפחות אחד האופרנדים הוא בעל ערך FALSE (0).

לפיכך, "אחד וגם אפס" יתנו אפס, וכמובן: "אפס וגם אפס" יתנו אפס, ורק "אחד וגם אחד" ישאירו את הערך אחד.

את הנאמר במילים אפשר לסכם גם ב"טבלת אמת" של הפקודה **AND**:

	0	**1**
0	0	0
1	0	1

מהו השימוש הפרקטי בפקודה **AND** ?

היא מאפשרת **לכבות** סיביות דולקות.

ועתה משימה.

■ בפרק ד לעיל, בנית תוכנה קצרה המקדמת את צבעי רקע המסך במהירות לפי תחלופת "שניות" המחשב.

עכשיו הגיע הזמן להאט את הקצב, וגם את מספר הצבעים המתחלפים!

כתוב תוכנה המחליפה 16 צבעים, במשך "דקת מחשב" אחת (כ-6 שניות).

לצורך המשימה השתמש בפקודה **AND**.

רמז: במהלך המנייה תוכל "להקפיא" כמה ביטים ולמנוע מהם להשתנות, ובאופן זה לגרום למעשה למונה המספרים להשתהות למשך פרקי זמן קבועים.

[תשובה 28 בעמוד 148]

■ הרץ את התוכנית והתבונן בצבעים הכהים המתחלפים.

שים לב – האוגר למעשה קיבל את הערכים הבאים:

0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208, 224, 240, 256

? האם זה מזכיר לך פקודת בייסיק?

אם חשבת על השימוש ב-**STEP** בלולאת ה-**FOR** ו-**NEXT** צדקת!

■ הוסף עתה שורת קוד נוספת שתהפוך את הצבעים המתחלפים לבהירים

(15, 31, 47, 63, 79, 95, ...)

[תשובה 29 בעמוד 149]

3. הפקודה EOR

הפקודה הלוגית **EOR** (קיצור המילים: Exclusive Or) **תיתן ערך** FALSE **(0) אם** שני האופרנדים זהים, וערך TRUE (1) אם שני האופרנדים שונים זה מזה.

לפיכך, "אחד או"מ אפס" יתנו אפס, וכמובן: "אפס או"מ אפס" יתנו אפס, ורק "אחד או"מ אחד" ישאירו את הערך אחד.

(הערה: השתמשתי ב"או"מ" כראשי תיבות למילים: "או מוציא" – התרגום העברי למונח: Exclusive Or)

את הנאמר במילים אפשר לסכם גם ב"טבלת אמת" של הפקודה **EOR**:

	0	**1**
0	0	1
1	1	0

מהו השימוש הפרקטי בפקודה **EOR** ?

היא מאפשרת **להפוך** סיביות (מסיבית דולקת לסיבית כבויה ולהיפך).

הפקודה **EOR** היא פקודה שימושית ביותר בתכנות, ועל מנת להדגים את אחד השימושים שלה ניעזר בכתובת הזיכרון 755 (או $2F3) האחראית על נראות הסמן (אותו ריבוע לבן המסמן היכן על המסך אנו נמצאים).

ערך ברירת המחדל השמור בבית $2F3 הוא 2.

■ פקוד לתוכו את הערך 0

(להזכירך, תוכל לעשות זאת אם תעבור למצב ה-DEBUG, באמצעות פקודת הכתבן **c**)

שים לב, הסמן נעלם ואינו מלווה אותך עוד כשאתה מקליד סימנים.

■ החזר את המצב לקדמותו ופקוד לתוך הבית $2F3 את הערך 2.

עתה, נניח שאנו רוצים להפוך את הסמן שלנו מסמן סטאטי לסמן מהבהב. נוכל לעשות זאת אם נדליק ונכבה לסרוגין את הסמן במרווחי זמן קבועים.

על מנת לעשות זאת נוכל להיעזר בשני המקרים הייחודיים של פעולת האו"מ (או מוציא):

(1) כאשר מפעילים או"מ של ערך עם עצמו (למשל 15 או"מ 15) נקבל תמיד אפס

(2) כאשר מפעילים או"מ של ערך עם אפס (למשל 100 או"מ 0) נקבל תמיד את אותו הערך ללא שינוי.

■ בדוק זאת, לפי טבלת האמת של הפקודה EOR !

כל זה מוביל אותנו למסקנה שאם ניקח ערך כלשהו ונפעיל עליו שוב ושוב את האו"מ עם אותו הערך, נקבל פעם את הערך עצמו ופעם את הערך אפס, לסרוגין.

לדוגמה: 2 או"מ 2, נותן 0. ואז: 0 או"מ 2 נותן 2, ושוב 2 או"מ 2 נותן 0 וחוזר חלילה.

■ כתוב עתה תוכנית קצרה שמדליק ומכבה לסרוגין את הסמן במרווחי זמן של כרבע שנייה.

[תשובה 30 בעמוד 149]

התוכנה שכתבת נותנת אפקט נחמד של הסמן. האם ניתן להפוך את הסמן למהבהב גם ללא תלות בתוכנה? כלומר להשאירו כזה גם כאשר נחזור אל מצב הכתבן ונמשיך להקליד שורות קוד נוספות?

התשובה היא כן, אך רק בספר הבא!

הפקודה BIT

אחרי שעברנו על שלושת הפקודות הלוגיות ראוי להציג פקודה נוספת מאותה משפחה, והיא הפקודה BIT.

הפקודה BIT (קיצור המילים: BIt Test) מאפשרת לבחון **סיבית בודדת אחת** בתוך כתובת זיכרון מסוימת, על ידי השוואה בין כתובת זו לבין תוכנו של האוגר.

על מנת לבצע את אותה בחינה, המעבד נעזר בעקרון פעולתה של הפקודה הלוגית AND ותוצאת הבחינה מיתרגמת למצבו של דגל השוויון Z.

ניקח למשל את כתובת הזיכרון $14 המונה כזכור את "שניות" המחשב, ואשר מעצם טיבה נושאת ערך מתחלף (בין 00000000 ל-11111111).

[שים לב, מכיוון שאנו עוסקים בתחום הפקודות הלוגיות, אנו עובדים בשיטת הספירה הבינארית!]

אם נרצה למשל לנטר בתוך כתובת הזיכרון הזו אך ורק את הסיבית החמישית, נוכל לכבות לרגע את כל הסיביות האחרות, זולת הסיבית החמישית ולבחון כיצד סיבית ספציפית זו מתנהגת.

לצורך כך כל שעלינו לעשות הוא להנגיד את הערך השמור בכתובת הזיכרון הנדונה עם המספר הבינארי: 00010000 (שבו הסיבית החמישית דולקת), ולהשתמש בעקרון הפעולה הלוגית: and

זכור כי רק המפגש בין שני אופרנדים בעלי ערך TRUE (1) ישאיר את הסיבית דולקת ודי באופרנד אחד בעל ערך FALSE (0) כדי לכבות את הסיבית.

נבחן ארבעה דוגמאות. לצורך הנוחות, צבעתי את הסיבית החמישית באדום:

א. 00010000 (השנייה ה-16)

מכיוון שהסיבית החמישית דולקת: 00010000 <= 00010000 [and] 00010000

הסיבית נשארת דולקת, וכך גם הדגל Z.

ב. 00100000 (השנייה ה-32)

מכיוון שהסיבית החמישית כבויה: 00000000 <= 00010000 [and] 00100000

הסיבית נכבית, וכך גם הדגל Z.

ג. 00110000 (השנייה ה-48)

מכיוון שהסיבית החמישית דולקת: 00010000 <= 00010000 [and] 00110000

הסיבית נשארת דולקת, וכך גם הדגל Z.

ד. 11111111 (השנייה ה-255)

מכיוון שהסיבית החמישית דולקת: 00010000 <= 00010000 [and] 11111111

הסיבית נשארת דולקת, וכך גם הדגל Z.

שים לב, בניגוד לפקודה **AND** הפקודה **BIT** אינה משנה כהוא זה, לא את הערך השמור באוגר ולא את הערך השמור בתא הזיכרון.

? כיצד אפוא נעזרים בה?

באמצעות הפקודות **BEQ** ו-**BNE** שכבר פגשנו בפרקים הקודמים – פקודות הבוחנות את מצבו של דגל האפס Z.

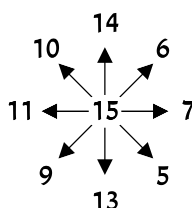
ועתה שתי משימות:

■ במשימה הראשונה, עליך להיעזר ב"שניות המחשב" ולהפוך את רקע המסך לסירנה מהבהבת, שתחליף צבעים בין כחול ואדום לסירוגין במרווחים של כרבע שנייה.
לצורך המשימה, השתמש בפקודה **BIT**

רמז: אם תרשום לפניך את כל הרצפים הבינאריים בין 00000000 ל-11111111 תגלה כי הסיבית החמישית נשארת כבויה ברצף 16 פעמים, אחר כך דולקת ברצף 16 פעמים וחוזר חלילה.

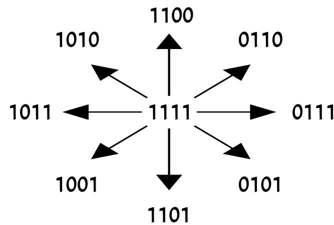
(תשובה 31 בעמ' 150)

■ במשימה השנייה נבנה תוכנית לבדיקת הג'ויסטיק!
אתה יודע כבר כי הבית המנטר את פעולת הג'ויסטיק הוא $278.
ואתה ודאי זוכר כי הערכים אשר הג'ויסטיק עשוי לקבל הם אלו:



האם חשבת פעם מה משמעות הערכים הללו?

ובכן דע לך כי לא מדובר בערכים שרירותיים, אלא יש הגיון מאחוריהם, אשר מתגלה מיד אם נמיר את הערכים הללו לשיטת הספירה הבינארית.



התבונן בשלושת הערכים הימניים: 0110, 0111, 0101.

?

האם אתה רואה את המכנה משותף ביניהם?

ובכן, ודאי שמת לב כי הסיבית הרביעית בשלושתם היא 0.

נסכם. ככל שהג'ויסטיק זז ימינה (לרבות ימינה ולמטה, או ימינה ולמעלה) – הסיבית הרביעית בערך הבינארי לעולם תהייה כבויה.

עתה, התבונן בשלושת הערכים השמאליים: 1010, 1011, 1001.

גם כאן ניתן לסכם. ככל שהג'ויסטיק זז שמאלה (וגם שמאלה ולמטה, או שמאלה ולמעלה) – הסיבית השלישית בערך הבינארי לעולם תהייה כבויה.

באותו אופן, גם לגבי הערכים העליונים והתחתונים נוכל לראות כי:

ככל שהג'ויסטיק זז למטה (וגם למטה וימינה, או למטה ושמאלה) – הסיבית השנייה בערך הבינארי לעולם תהייה כבויה.

ככל שהג'ויסטיק זז למעלה (וגם למעלה וימינה, או למעלה ושמאלה) – הסיבית הראשונה בערך הבינארי לעולם תהייה כבויה.

■ עתה, כשאתה מצויד בידע הזה על אודות הג'ויסטיק, נסה לכתוב בעצמך תוכנה קצרה שתדליק את הסימן "●" באחד מארבעת הכיוונים (ימין, שמאל, למטה, למעלה), לפי תזוזת הג'ויסטיק.

בכל מקרה, עיין גם בהסבר המפורט להלן.

בשלב הראשון נבנה את הלולאה המרכזית המנטרת את מצבו של הגויסטיק. כמו כן נוסיף פקודת דילוג רגילה מן הלולאה אם הג'ויסטיק נמצא במצב מנוחה (IDLE). במצב זה, נדאג לאפס את ארבעת הנקודות הרלוונטיות על המסך (ימינה, שמאלה, למעלה, למטה).

```
10    *=$600
20  LOOP
30    LDA  $278
40    CMP  #$F
50    BEQ  CLR
0260  FIN
0270    JMP  LOOP
0280  CLR
0290    LDX  #$0
0300    STX  $9C5E
0310    STX  $9C84
0320    STX  $9C88
0330    STX  $9CAE
0340    JMP  FIN
```

אחר כך נוסיף ארבע בדיקות – אחת כנגד כל כיוון, תוך שימוש בפקודה BIT.

נפתח בכיוון למעלה ונבחן את הסיבית הראשונה באמצעות הפקודה **BIT**.
כל עוד הסיבית הראשונה **אינה** נדלקת, משמע שהג'ויסטיק לא נע כלפי מעלה. ואם זה המצב, נוכל לדלג הלאה.
באותו אופן נבחן את הסיבית השנייה, עבור הכיוון מטה; נדלג ונבחן את הסיבית השלישית, עבור הכיוון שמאלה, ולבסוף נדלג ונבחן את הסיבית הרביעית, עבור הכיוון ימינה.

```
60    LDX #$60
70    LDA #$1
80    BIT $278
90    BNE DOWN
0100     STX $9C5E
0110  DOWN
0120     LDA #$2
0130     BIT $278
0140     BNE LEFT
0150     STX $9CAE
0160  LEFT
0170     LDA #$4
0180     BIT $278
0190     BNE RIGHT
0200     STX $9C84
0210  RIGHT
0220     LDA #$8
0230     BIT $278
0240     BNE FIN
0250     STX $9C88
```

עד כה ראינו כי הפקודה **BIT** משפיעה על דגל השוויון Z, ומאפשרת לבחון באמצעותו אם סיבית מסוימת דלוקה או מכובה.

דע לך כי הפקודה **BIT** משפיעה על שני דגלים נוספים: דגל השלילה N ודגל הגלישה V:

כאשר אנו מפעילים את הפקודה **BIT** על תא זיכרון מסוים – ובלי קשר לערך השמור באוגר – ככל שהסיבית השמינית בתא הזיכרון דולקת, דגל השלילה N יידלק, וככל שהסיבית השביעית בתא הזיכרון דולקת, דגל הגלישה V יידלק.

במסגרת יחידת לימוד זו לא אכנס לשאלה **מדוע** קיים מאפיין שכזה בפקודה **BIT**, ואולם בהחלט נוכל להשתמש בו לצרכינו התכנותיים.

■ זוכר את ה"סירנה" שבנית? עתה אבקשך להכניס שינוי בתוכנה – להפוך את הסירנה לאיטית יותר, כך שקצב תחלופתה מבוסס על השינוי בסיבית השביעית של הבית המונה את "שניות המחשב".

[תשובה 32 בעמ' 150]

■ ערוך עוד שינוי בתוכנה על מנת להפוך את הסירנה לאיטית במיוחד, כך שקצב התחלופה מבוסס על השינוי בסיבית השמינית.

[תשובה 33 בעמ' 150]

פרק י"ב
בחזרה לבייסיק

בפרק המבוא הצגתי כדוגמה את הרצף הבינארי הזה:

1010100100100010100011011100100000000010

וטענתי כי הרצף המדויק הזה – ככל שיימסר למעבד המחשב שלך – יגרום לו לצבוע את רקע מסך הטלביזיה באדום!

עתה הגיע הזמן לבדוק את נכונות הטענה!

ראשית, ניקח את הרצף ונחלק אותו למקטעים של שמונה סיביות:

10101001 00100010 10001101 11001000 00000010

אם נמיר את הרצפים הללו מבסיס בינארי לבסיס הקסה-דצימלי נקבל:

$A9, $22, $8D, $C8, $2

לצורך הנוחיות נצבע את הערכים הללו בשני צבעים:

$A9, $22, $8D, $C8, $2

? האם הערכים הצבועים באדום נראים לך מוכרים?

שים לב, אם נצמיד את שני הערכים האחרונים זה לזה נקבל: $2C8 (או בשיטה העשרונית: 712). ואם נכניס אל תוך בית 712 את הערך 34 (או: $22 בשיטה ההקסה-דצימלית), רקע המסך ייצבע באדום!

? האם תוכל לנחש עתה מה מסמלים הערכים הצבועים בירוק?

ובכן הערך $A9 שקול לפקודת האסמבלי: # LDA (ככל שפקודה זו מצפה לערך כשלעצמו).

ואילו הערך $8D שקול לפקודת האסמבלי: STA (ככל שפקודה זו מציבה את ערכו של האוגר בכתובת זיכרון הגבוהה מ-256).

במובן זה, כל אחת מן הפקודות LDA ו-STA היא למעשה "עזר זיכרון", או: "נמוניק" (NEMONIC), עבור הקוד המספרי שנמסר למחשב. בהערת אגב, אציין כי המילה "נמוניק" נגזרת מהשם "נמוזין" – שמה של אלת הזיכרון היוונית.

אינך חייב לזכור, אפוא, את ייצוגיהן המספריים של פקודות האסמבלי (אם כי לא יזיק לשנן כמה מרכזיות). כתבן האסמבלר עושה את מלאכת התרגום בשבילך.

יחד עם זאת, על מנת לשלב בתוכנות הבייסיק שלך רוטינות באסמבלי – עליך למסור למחשב את הערכים המספריים עצמם – את אותם ערכים אשר מעבד האסמבלר הפיק בעבורך.

? באיזה בסיס ספירה עליך לעבוד?

כמובן – בסיס הספירה העשרוני! זוהי סביבת העבודה של בייסיק, לטוב ולרע.

נחזור לדוגמה שבה פתחנו. אם תרצה לשלב בתוך תוכנת הבייסיק שלך את הרוטינה הצובעת את המסך באדום, עליך למסור למחשב את הרצף העשרוני הבא: 169, 34, 141, 200, 2

? כיצד תמסור את המידע הזה למחשב?

כמובן, באמצעות פקודת הבייסיק: **POKE**

את הרוטינה שכתבת (קרי: את אותו רצף של ערכים עשרוניים) תוכל לכתוב באותו מקום בזיכרון בו השתמשת כשעבדת באסמבלר – הלוא הוא הדף השישי (החל מבית $600 או בשיטת הספירה העשרונית: 1536).

והשאלה החשובה ביותר...

? כיצד נודיע למחשב לצאת לרגע מתוכנת הבייסיק ולהפעיל את אותה רוטינה שכתבנו בשפת מכונה?

על כך בסעיף הבא, אך ראשית כל, הערה.

על מנת שתוכנת הבייסיק תוכל לגשת לרוטינה בשפת מכונה חייבים לטעון את המחסנית בנתון אחד לפחות (אפילו את הערך: 0). כמו כן, חייבים להודיע למחשב מתי מסתיימת הרוטינה ועליו לשוב אל תוכנית הבייסיק.

כדי לעמוד בשני התנאים הללו, נקפיד לפתוח כל רוטינה (המיועדת לפעול מתוך בייסיק) בפקודה **PLA**, ולסיים אותה בפקודה: **RTS**.

הפקודה **PLA** מקבילה לערך ההקסה-דצימלי $68, או 104 בשיטה העשרונית, ואילו הפקודה **RTS** מקבילה לערך ההקסה-דצימלי $60 (או 96 בשיטה העשרונית).

לכן על מנת להפעיל את הרוטינה מתוך תוכנת הבייסיק, נכניס החל מבית 1536 ועד לבית 1542 את הערכים הללו: 104, 169, 34, 141, 200, 2, 96.

פקודת הבייסיק USR

פקודת הבייסיק המשמשת קישור אל רוטינות בשפת מכונה היא הפקודה:
USR

השימוש בפקודה זו פשוט ביותר. כל שעלינו לעשות הוא לבחור משתנה דמי כלשהו (כלומר משתנה שאין לו שום משמעות בתוכנת הבייסיק), וצורת כתיבת הפקודה היא:

U=USR(1536)

שים לב, המספר המופיע בתוך הסוגריים הוא כתובת הזיכרון הראשונה בה יושבת הרוטינה שכתבנו, כמובן לפי שיטת הספירה העשרונית.

■ הוצא כעת את הקרטריג׳ של האסמבלר (אל תשכח לשמור תוכנות שעבדת עליהן) ועבור לסביבת הבייסיק.

כתוב תוכנת בייסיק אשר כותבת לתוך הזיכרון את ערכי הרוטינה שיצרנו לעיל (צביעת רקע המסך באדום), ואחר כך מפעילה את אותה רוטינה.

לצורך הכנסת ערכי הרוטינה בזה אחר זה למקומם הייעודי, תוכל להשתמש בזוג פקודות הבייסיק: **READ** ו-**DATA**

[תשובה 34 בעמוד 151]

ונסיים ב... לא לעשות כלום

הפקודה האחרונה שאסקור ביחידת לימוד זו היא פקודה ש... לא עושה כלום! אך לפני כן, הסבר קצר.

אתה כבר יודע כי מעבד ה-6502 של מחשב האטארי שלך מבצע למעלה מ-1500 פעולות בשנייה, כלומר בכל רגע נתון הוא מביא פקודה מתוך כתובת מסוימת בזיכרון המחשב; מפענח אותה על מנת "להבין" מה היא צריכה לבצע, ואחר כך ניגש לבצע אותה על נתונים שונים במחשב.

הרצף הזה של "הבאה–פיענוח–ביצוע" נקרא "מחזור" (Cycle). ולמעשה מרגע הדלקת המחשב ועד לכיבויו עובר המחשב רבבות, ואף מיליוני מחזורים שכאלה.

האם אפשר להורות למעבד לבצע מחזור ריק? כלומר להביא לו פקודה שהפיענוח שלה הוא לא לעשות מאומה?

התשובה היא כן, וזו בדיוק מהות הפקודה **NOP**
(קיצור המילים: No OPeration)

? אבל רגע אחד. למה לכל הרוחות ולכל הסיביות, נדרשת בכלל פקודה שכזו?

ובכן, אחד השימושים בפקודה **NOP** הוא לסנכרן בין פעולתם של שני רכיבים במחשב (למשל המעבד המרכזי וה-ANTIC). בנושא זה נעסוק בהרחבה ביחידת הלימוד הבאה...

שימוש מסוג שונה לחלוטין שתוכל לעשות בפקודה **NOP** הוא לסמן לעצמך שורות קוד שתידרש להשלים בהמשך. כאשר המחשב יפגוש בשורת קוד המכילה את ההוראה **NOP** הוא פשוט ימשיך הלאה.

זה המקום להציג עוד עזר תכנותי שיכול לשמש אותך, וזוהי פקודת הסימון: **;**

בכל אימת שכתבן האסמבלר יפגוש בפקודה הזו, הוא יתעלם מכל מה שכתוב בהמשך אותה שורת קוד.

במובן זה הפקודה "**;**" מזכירה מאוד את פקודת הבייסיק: **REM**.

סיכום
אז מה היה לנו שם?

ביחידת לימוד זו עברנו על רוב רובן של פקודות האסמבלי למעבד 6502 – וליתר דיוק, על 55 מתוך 56 הפקודות הקיימות!

להלן רשימה של כל פקודות השפה, בסדר אלפביתי. לצד כל פקודה הסבר קצר, והפניה לעמודים הרלוונטיים בחוברת זו.

<table dir="rtl">
<tr><th>הפקודה</th><th>הסבר קצר</th><th>עמודים</th></tr>
<tr><td>ADC</td><td>חבר עם שארית (פועלת על האוגר בלבד).
<code>ADC #$1A</code></td><td>26-33</td></tr>
<tr><td>AND</td><td>פקודה לוגית המשמשת לכיבוי סיביות דולקות.
טבלת האמת שלה היא:
<table dir="ltr"><tr><th></th><th>0</th><th>1</th></tr><tr><th>0</th><td>0</td><td>0</td></tr><tr><th>1</th><td>0</td><td>1</td></tr></table><code>AND #$1A</code></td><td>112-113</td></tr>
<tr><td>ASL</td><td>הסטה שמאלה של כל הסיביות. משמשת להכפלת ערך פי 2. (פועלת על האוגר בלבד)
<code>ASL A</code></td><td>99-103</td></tr>
<tr><td>BCC</td><td>דילוג מלולאה אם דגל השארית (C) כבוי.
<code>BCC MYLABEL</code></td><td>47</td></tr>
<tr><td>BCS</td><td>דילוג מלולאה אם דגל השארית (C) דולק.
<code>BCS MYLABEL</code></td><td>46-47</td></tr>
</table>

BEQ	דילוג מלולאה אם דגל השוויון (Z) דולק. BEQ MYLABEL	42-43
BIT	בדיקת סיבית בודדת בתוך תא זיכרון מול האוגר. תוצאת הבדיקה נשמרת לתוך דגל השוויון (Z). הסיבית השמינית נשמרת לתוך דגל השלילה (N) והסיבית השביעית נשמרת לתוך דגל הגלישה (V) BIT $278	116-122
BMI	דילוג מלולאה אם דגל השלילה (N) דולק BMI MYLABEL	66
BNE	דילוג מלולאה אם דגל השוויון (Z) כבוי BNE MYLABEL	43-44
BPL	דילוג מלולאה אם דגל השלילה (N) כבוי BPL MYLABEL	66
BRK	עצירת פעולתה של התוכנה BRK	91-92
BVC	דילוג מלולאה אם דגל הגלישה (V) כבוי BVC MYLABEL	70
BVS	דילוג מלולאה אם דגל הגלישה (V) דולק BVS MYLABEL	70
CLC	כיבוי דגל השארית C. CLC	31-33
CLD	כיבוי הדגל העשרוני D. CLD	93
CLI	כיבוי דגל ההפרעה I. CLI	89
CLV	כיבוי דגל הגלישה V. CLV	70

CMP	השוואת ערכו של האוגר לערך מסוים. CMP #$FF	41
CPX	השוואת ערכו של רגיסטר X לערך מסוים. CPX #$FF	50
CPY	השוואת ערכו של רגיסטר Y לערך מסוים. CPY #$FF	50
DEC	הקטנת ערכו של תא זיכרון ביחידה אחת. DEC $9C40	55-57
DEX	הקטנת ערך רגיסטר X ביחידה אחת. DEX	50-53
DEY	הקטנת ערך רגיסטר Y ביחידה אחת. DEY	50-53
EOR	פקודה לוגית המשמשת להפיכת סיביות. טבלת האמת שלה היא: 0 1 0 0 1 1 1 0 EOR #$1A	114-116
INC	הגדלת ערכו של תא זיכרון ביחידה אחת. INC $9C40	55-57
INX	הגדלת ערך רגיסטר X ביחידה אחת. INX	50-53
INY	הגדלת ערך רגיסטר Y ביחידה אחת. INY	50-53
JMP	קפיצה (למשל אל תווית). JMP MYLABEL	39-41

JSR	קפיצה לסאב רוטינה (עם אפשרות חזרה). JSR MYLABEL	60-61
LDA	טעינת ערך לתוך האוגר. ערך כשלעצמו LDA #$90 ערכו של תא זיכרון LDA $2F4 ערך משתנה LDA MYDATA,X	20-22
LDX	טעינת ערך לתוך רגיסטר X ערך כשלעצמו LDX #$90 ערכו של תא זיכרון LDX $2F4	49
LDY	טעינת ערך לתוך רגיסטר Y ערך כשלעצמו LDY #$90 ערכו של תא זיכרון LDY $2F4	49
LSR	הסטה ימינה של כל הסיביות משמשת לחלוקת ערך ב-2. (פועלת על האוגר בלבד) LSR A	104-105
NOP	פקודה שאינה מבצעת דבר. NOP	126-127
ORA	פקודה לוגית המשמשת להדלקת סיביות כבויות. טבלת האמת שלה היא: (טבלה: שורות/עמודות 0,1 — 0 OR 0 = 0, 0 OR 1 = 1, 1 OR 0 = 1, 1 OR 1 = 1) ORA #$1A	110-112
PHA	העתקת הערך השמור באוגר אל תוך המחסנית. PHA	77-78
PHP	העתקת "תמונת מצב" הדגלים אל תוך המחסנית. PHP	97-98

	0	1
0	0	1
1	1	1

PLA	שליפת הערך השמור במחסנית וטעינתו לתוך האוגר. PLA	77-78
PLP	סימון כל הדגלים לפי "תמונת המצב" שנשמרה במחסנית. PLP	97-98
ROL	סיבוב כל סיביות המספר שמאלה. (פועלת על האוגר בלבד) ROL A	107-109
ROR	סיבוב כל סיביות המספר ימינה. (פועלת על האוגר בלבד) ROR A	107-109
RTI	חזרה מהפרעה מסוג NMI RTI	לא נלמד
RTS	חזרה מסאב רוטינה. RTS	60-61
SBC	חסר עם שארית (פועלת על האוגר בלבד). SBC #$1A	33-37
SEC	הדלקת דגל השארית C. SEC	34
SED	הדלקת הדגל העשרוני D. SED	93
SEI	הדלקת דגל ההפרעה I. SEI	89
STA	הצבת הערך השמור באוגר לתוך תא זיכרון. תא יחיד STA $9C40 סדרת תאים STA $9C40,X	22

STX	הצבת הערך השמור ברגיסטר X לתוך תא זיכרון. STX $9C40	50
STY	הצבת הערך השמור ברגיסטר Y לתוך תא זיכרון. STY $9C40	50
TAX	העתקת הערך השמור באוגר לתוך רגיסטר X. TAX	57-58
TAY	העתקת הערך השמור באוגר לתוך רגיסטר Y. TAY	57-58
TSX	שמירת "מחוון המחסנית" ברגיסטר X. TSX	79-80
TXA	העתקת הערך השמור ברגיסטר X לתוך האוגר. TXA	57-58
TXS	קביעת "מחוון המחסנית" לפי הערך השמור ברגיסטר X. TXS	79-80
TYA	העתקת הערך השמור ברגיסטר Y אל האוגר. TYA	57-58

נספח
שיטות ספירה

שיטת הספירה הטבעית של בני אדם היא השיטה העשרונית, ואין בכך פלא. כל ילד לומד להכיר את עשר אצבעותיו וגם לספור באמצעותן.

ומה קורה כשנגמרות האצבעות ורוצים להמשיך ולספור הלאה? פשוט מאוד, אנו מקצים מקום (בין אם על הנייר ובין אם בזיכרוננו) לספרה נוספת ומייצגים באמצעותה את ספרת העשרות. כך גם לגבי ספרת המאות, האלפים וכן הלאה.

נתבונן למשל על המספר העשרוני: 456

הספרה 6 מייצגת כמובן את ספרת האחדות, הספרה 5 – את ספרת העשרות ואילו הספרה 4, את ספרת המאות.

נוכל לפרק את המספר ולכתוב אותו גם בצורה הזו:

$$456= 4*100 + 5*10 + 6*1$$

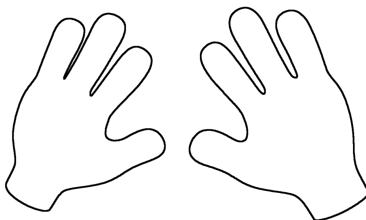
שים לב, כשאנו עובדים בבסיס הספירה העשרוני, משתמעים מכך שני דברים: ראשית, אנו נעזרים בעשר ספרות (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) ושנית, מקומה של כל ספרה מיוצג לפי כפולות של עשר.

נוכל להציג זאת גם באמצעות שימוש בחזקה:

$$456= 4*10^2 + 5*10^1 + 6*10^0$$

בתצוגה הזו, קל לראות כי החזקה מציינת גם את מיקומה של כל ספרה במספר: הספרה 4 נמצאת 2 מקומות שמאלה מנקודת ההתחלה, הספרה 5 נמצאת מקום אחד שמאלה מנקודת ההתחלה, ואילו הספרה 6 נמצאת אפס מקומות מנקודת ההתחלה.

עתה, נדמה בנפשנו כי נבראנו לא עם עשר אצבעות, אלא עם שמונה (כמו הדמויות בסרטים המצוירים). בעולם שכזה סביר להניח שהיינו משתמשים בבסיס הספירה האוקטלי, כלומר בסיס 8.



הספרות שהיו עומדות אז לרשותנו היו שמונה במספר (0,1,2,3,4,5,6,7)

ואילו מקומה של כל ספרה היה מיוצג לפי כפולות של 8

המספר האוקטלי: 456_8, למשל, היה נכתב כך:

$456_8 = 4*8^2 + 5*8^1 + 6*8^0$

(הערה: הספרה 8 המופיעה בכתב תחתי מסמלת את בסיס הספירה האוקטלי)

שים לב כי גם במקרה הזה, הספרה 4 נמצאת 2 מקומות שמאלה, הספרה 5 נמצאת מקום אחד שמאלה, ואילו הספרה 6 נמצאת על נקודת ההתחלה.

ואולם, ערכו של המספר האוקטלי 456_8 שונה לחלוטין מערכו של המספר העשרוני 456_{10}.

אם נחשב (בחישוב עשרוני) את תרגיל החזקות האחרון נקבל:

$4*8^2 + 5*8^1 + 6*8^0 = 256+40+6 = 302$

שפת התכנות אסמבלי משתמשת כאמור בשיטת הספירה ההקסה-דצימלית. ממש כמו בכל שיטת ספירה אחרת, שיטת הספירה ההקסה-דצימלית משתמשת ב-16 ספרות (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F), ומקומה של כל כל ספרה מיוצג לפי כפולות של 16.

נתבונן עתה על המספר ההקסה-דצימלי: 456_{16}.
נכתוב את תרגיל החזקות:

$456_{16} = 4*16^2 + 5*16^1 + 6*16^0$

נחשב (בחישוב עשרוני) את התרגיל ונקבל:

$456_{16} = 1024+80+6 = 1110$

ומה לגבי המספר ההקסה-דצימלי: $3E8_{16}$?

גם הפעם נפרק אותו כפי שלמדנו. במקום הספרה E נציב כמובן את ערכה העשרוני 14, ונקבל:

$3E8_{16} = 3*16^2 + 14*16^1 + 8*16^0 = 768+224+8 = 1000$

עד כאן בעניין שיטות הספירה.

אני מציע לך למצוא ברשת האינטרנט מחשבון שממיר בקלות מספרים משיטת ספירה אחת לשנייה.

תשובות

[תשובה 1 מעמ' 21]

הפקודה בשורה 20 תציב באוגר את הערך: $18 ואילו הפקודה בשורה 10 תציב באוגר את הנתון השמור בתא הזיכרון $18.

[תשובה 2 מעמ' 23]

```
10    *=$600
20    LDA #$0
30    STA $2C6
```

[תשובה 3 מעמ' 32]

```
10    *=$600
20    LDA #$11
30    STA $2C6
40    CLC
50    ADC #$E
60    STA $2C8
```

[תשובה 4 מעמ' 37]

```
10    *=$600
20    LDA #$2F
30    STA $9C40
```

```
40    SEC
50    SBC #$1
60    STA $9C41
```

[תשובה 5 מעמ' 44]

```
10    *=$600
20 LOOP LDA $278
30    CMP #$F
40    STA $9C40
50    BNE EXIT
60    JMP LOOP
70 EXIT
```

[תשובה 6 מעמ' 47]

אלו הן השורות שיש להוסיף:

```
60    CMP #$20
70    BCS EXIT
90 EXIT
```

[תשובה 7 מעמ' 48]

אלו הן השורות שיש להחליף (לדרוס):

```
20    LDA #$FF
60    CMP #$2F
70    BCC EXIT
```

[תשובה 8 מעמ' 52]

אלו הן השורות שיש להוסיף או להחליף (לדרוס):

```
50      LDY #$0
60      STY $14
70 TIMER LDY $14
80      CPY #$20
90      BEQ CONT
100      JMP TIMER
110 CONT JMP LOOP
```

[תשובה 9 מעמ' 53]

אלו הן השורות שיש להחליף (לדרוס):

```
20      LDY #$FF
30 LOOP DEY
40      STY $2C8
50      LDX #$0
60      STX $14
70 TIMER LDX $14
80      CPX #$5
```

[תשובה 10 מעמ' 54]

```
35    BEQ  EXIT
120  EXIT
```

[תשובה 11 מעמ' 55]

```
10    *=$600
20    LDX  #$0
30  LOOP  INX
40    CPX  $FF
50    BEQ EXIT
60    LDA  #$E
70    STA  $9C40,X
80    JMP  LOOP
90  EXIT
```

[תשובה 12 מעמ' 55]

אלו הן השורות שיש להוסיף או להחליף (לדרוס):

```
15    LDA  #$0
60    ADC  #$1
```

[תשובה 13 מעמ' 66]

```
10    *=$600
20 LOOP LDA $14
30    BMI MINUS
40    BPL PLUS
50    JMP LOOP
60 MINUS LDX #$D
70    STX $9C40
80    JMP LOOP
90 PLUS LDX #$B
100    STX $9C40
110    JMP LOOP
```

[תשובה 14 מעמ' 74]

אלו הן השורות שיש להוסיף או להחליף (לדרוס):

```
30    STX $6FE
100    CPX $6FE
180    LDX $2FC
190    STX $6FE
```

[תשובה 15 מעמ' 78]

```
10    *=$600
20    LDA #$72
30    PHA
40    LDA #$61
50    PHA
60    LDA #$74
70    PHA
80    LDA #$73
90    PHA
110    PLA
120    STA $9C40
130    PLA
140    STA $9C41
150    PLA
160    STA $9C42
170    PLA
180    STA $9C43
```

[תשובה 16 מעמ' 79]

אלו הן השורות שיש להוסיף:

```
100    TSX
190    TXS
200    PLA
210    STA $9C45
220    PLA
```

```
230     STA $9C46
240     PLA
250     STA $9C47
260     PLA
270     STA $9C48
```

[תשובה 17 מעמ' 85]

```
10    *=$600
20    LDX #$0
30 LOOP INX
40    LDA $E000,X
50    STA $9800,X
60    LDA $E100,X
70    STA $9900,X
80    LDA $E200,X
90    STA $9A00,X
100     LDA $E300,X
110     STA $9B00,X
120     CPX #$FF
130     BEQ EXIT
140     JMP LOOP
150 EXIT
160     LDA #$98
170     STA $2F4
```

[תשובה 18 מעמ' 86]

אלו הן השורות שיש להוסיף:

```
180    LDY #$0
200 LOOP2 LDA DATA,Y
210    CMP #$FF
220    BEQ END
230    STA $9B08,Y
240    INY
250    JMP LOOP2
260 DATA
270    .BYTE $0, $92, $D2, $E6, $CC, $FC, $0,
$0
280    .BYTE $C0, $FC, $6, $6, $E, $3C, $0, $0
290    .BYTE $0, $3C, $1C, $C, $C, $C, $0, $0
300    .BYTE $0, $7E, $66, $66, $66, $7E, $0,
$0
310    .BYTE $FF
320 END
```

[תשובה 19 מעמ' 89]

```
10    *=$600
20 LOOP LDA $2FC
30    STA $2C8
40    LDX $14
50    CPX #$80
```

```
60     BCS  IDLE
70     CLI
80     JMP  LOOP
90  IDLE  SEI
100     JMP  LOOP
```

[תשובה 20 מעמ' 92]

זוהי השורה שיש להוסיף:

```
55     BRK
```

[תשובה 21 מעמ' 98]

אלו הן השורות שיש להוסיף:

```
20     PHP
50     PLP
```

[תשובה 22 מעמ' 102]

```
10     *=$600
20     LDA  #$8
30     ASL  A
40  TEMP=0
50     STA  TEMP
60     ASL  A
70     ASL  A
80     ADC  TEMP
```

[תשובה 23 מעמ' 105]

```
10    *=$600
20 LOOP
30    LDA $13
40    LDX $14
50    LSR A
60    BCS LOOP
70    STX $2C8
80    JMP LOOP
```

[תשובה 24 מעמ' 105]

אלו הן השורות שיש להוסיף:

```
180      LDA  #$3
190  MOVE
200      STA  $9B08
210      ROL  A
300      LDY  #$0
310      STY  $14
320      JSR  TIMER
330      JMP  MOVE
400  TIMER LDY  $14
410      CPY  #$F
420      BEQ  TERM
430      JMP  TIMER
440  TREM RTS
```

[תשובה 25 מעמ' 108]

אלו הן השורות שיש להוסיף:

```
315     PHP
325     PLP
```

[תשובה 26 מעמ' 109]

אלו הן השורות שיש להוסיף:

```
215     STA  $9B09
220     ROL  A
225     STA  $9B0A
230     ROL  A
235     STA  $9B0B
240     ROL  A
245     STA  $9B0C
250     ROL  A
255     STA  $9B0D
260     ROL  A
265     STA  $9B0E
270     ROL  A
275     STA  $9B0F
280     ROL  A
285     ROL  A
290     ROL  A
295     ROL  A
```

[תשובה 27 מעמ' 112]

```
10    *=$600
20  START
30    LDX #$0
40  LEN
50    LDA TEXT,X
60    LDY $14
70    CPY #$80
80    BCC NEGATIVE
90  CONT
100   CMP #$FF
110   BEQ START
120   STA $9C40,X
130   INX
140   JMP LEN
200 TEXT
210   .BYTE "hello", $FF
300 NEGATIVE
310   ORA #$80
320   JMP CONT
```

[תשובה 28 מעמ' 113]

```
10    *=$600
20  LOOP
30    LDA $14
40    AND #$E0
50    STA $2C8
60    JMP LOOP
```

[תשובה 29 מעמ' 113]

זוהי השורה שיש להוסיף:

```
45    ORA #$E
```

[תשובה 30 מעמ' 115]

```
10    *=$600
20  CRSR
30    LDA $2F3
40    EOR #$2
50    STA $2F3
60    JMP TIMER
70    JMP CRSR
100 TIMER
110   LDX $0
120   STX $14
130 LOOP
140   LDX $14
150   CPX $#F
160   BEQ END
170   JMP LOOP
180 END
190   RTS
```

[תשובה 31 מעמ' 118]

```
10     *=$600
20     LDA  #$10
30  LOOP
40     BIT  $14
50     BEQ  RED
60     LDX  #$94
70     STX  $2C8
80     JMP  LOOP
90  RED
100     LDX  #$25
110     STX  $2C8
120     JMP  LOOP
```

[תשובה 32 מעמ' 122]

אלו הן השורות שיש להחליף (לדרוס):

```
30     LDA  #$0
50     BVS  RED
```

[תשובה 33 מעמ' 122]

זוהי השורה שיש להחליף (לדרוס):

```
50    BPL RED
```

[תשובה 34 מעמ' 126]

```
10 FOR X=0 TO 6
20 READ A
30 POKE 1536+X,A
40 NEXT X
50 U=USR(1536)
60 DATA 104,169,34,141,200,2,96
```

יחידת לימוד זאת פותחת לפניך עולם חדש של תכנות – תכנות בשפת אסמבלי.
בשפה פשוטה וקולחת מוצגות בזו אחר זו כל פקודות השפה, בליווי דוגמאות ותרגולים. עם סיום יחידת לימוד זו, תוכל לפתח בעצמך רוטינות בשפת מכונה, לשלבן בתוכניות בייסיק שתכתוב ולהרחיב את טווח יכולותיך התכנותיות.

נכתב ע"י: נדב שטכמן פולישוק